UiO : Department of Informatics
University of Oslo

Multi-Objective Optimisation of Stereo Vision Algorithms

Stian Selbek

Master's Thesis Autumn 2015

Multi-Objective Optimisation of Stereo Vision Algorithms

Stian Selbek

3rd August 2015

Abstract

The process of stereo vision matches one or more images to recover the depth information of the pictured scene. Great progress is currently being made within this field, with algorithmic research, computational power developments, and cheaper cameras all contributing to give stereo vision great future potential as the depth measuring system of choice. One of the challenges of the stereo vision approach is the multitude of control parameters, which all affect algorithm behaviour. These parameters have traditionally been tuned by hand, with some limited use of computerized optimisation techniques. However, the process of evolutionary computation provides a promising method of optimisation of such complex problems.

This thesis explored the possibility of applying a multi-objective evolutionary optimisation approach to the stereo algorithm parameter problem. In this regard, an automatic parameter optimisation framework based on the multi-objective optimisation algorithm NSGA-II was developed and tested.

In order to judge the performance of the framework and the validity of the multi-objective approach, three different stereo algorithms were tested and a series of near pareto-optimal parameter sets were produced. One parameter set per algorithm was submitted to the official KITTI stereo vision benchmark ranking, and was able to improve upon the current official results.

Acknowledgements

I wish to thank my two supervisors, Kyrre Glette and Johannes Solhusvik for their inspiration and help. With your assistance I was able to explore a new and exiting field.

I would also want to thank Andreas Geiger of MPI Tübingen for his assistance with the KITTI benchmark submission process.

Lastly, I want to thank my fellow students, family, friends, and the staff at the ROBIN group for their support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of computer stereo vision allows 3D depth information to be extracted from two or more partially overlapping images. This technique has found use in many areas including robotic exploration [21], planetary [57] and solar observation [37], autonomous cars and driver assistance applications [56], and video conferencing, among others. Yet rapid development in stereo algorithmic research, cheap consumer cameras and the computational power required to calculate the results, all point to the even greater potential inherent to this technique in the years to come.

However, a common issue with stereo algorithms is the difficult tuning of their algorithm control parameters, which all modify their specific behaviour. These parameters are typically tuned by hand by altering them in turn, thereby garnering a better idea of their effect on the resulting output. This is a slow and laborious process and that makes it difficult to account for both multiple parameters at once, and their possible inter-dependence and effect. Fast, near real-time stereo algorithms present the practical opportunity to use computational optimisation methods to automatically tune algorithm control parameters to maximise quality on example images over the course of thousands of evaluations. Some work has been done in this regard using common statistical optimisation methods [85], but these tend to require a representative statistical model for minimization. Other alternatives include grid search, which performs a limited exhaustive search, and is thus a computationally expensive technique.

Evolutionary algorithms, inspired by the natural process of evolution, may provide a promising possibility for stereo parameter optimisation. This optimisation strategy is able to find near-optimal solutions with comparatively simple steps, even in large unknown problems. Some examples include the evolution of virtual creatures [70], component design [9], job scheduling [7], path planning [35], or even stereo vision [22]. However, only limited work has been published in this area in regards to actual stereo parameter optimisation, and then only with an optimisation of a single quality objective [53, 73].

1.1 Goals

With the increased popularity of multi-objective optimisation techniques, this presents a good opportunity to test the possibility of using such a method on the computer stereo vision parameter optimisation problem. As such, the first goal of this thesis will be the implementation of a robust framework capable of parameter optimisation of stereo algorithms.

As a second goal, the possibility of using a multi-objective approach for algorithm parameter optimisation shall be investigated. This entails optimising several stereo algorithms to see if the multi-objective focus provides a feasible method for optimisation. Of particular secondary interest is the promising ability of a multi-objective approach to not only adapt parameters to different trade-offs, but also to compare stereo algorithm performance along their entire pareto fronts, giving the possibility of a more detailed view than traditional stereo algorithm comparisons.

1.2 Thesis Outline

This thesis is divided into five chapters with each dedicated to a certain aspect of the project. Beyond this introduction, the thesis starts with relevant background material presented in Chapter 2. This gives an overview of the computer stereo vision process, its different methods and certain algorithms relevant to the thesis. Background information in evolutionary algorithms with a focus on genetic algorithms and multi-objective optimisation is also covered in this chapter.

Chapter 3 describes the implementation of the stereo parameter optimisation framework and how its various modules operate. In Chapter 4 the experiments done to tune the details of the framework for more robust results is first presented. Then the framework is used to explore the possibility of improving on current benchmark results. Lastly Chapter 5 discusses the performance of the framework and the results acquired through its use. This chapter also contains the conclusion and a presentation of possible future work.

Chapter 2

Background

This chapter contains relevant background information to the thesis. First, Section 2.1 provides an introduction to stereo vision with a focus on the faster variants most relevant for use in evolutionary computing. A sample of stereo algorithms is described in more detail in Section 2.2, in addition to a review of how algorithms may be compared. Then an introduction to the concepts of evolutionary computing with a focus on genetic algorithms is done in Section 2.3. Lastly tools used in this thesis are presented in Section 2.4.

2.1 Stereo Vision

Stereo vision is the art of extracting depth from images by comparing two or more partially overlapping pictures taken from slightly different perspectives. The typical binocular setup takes inspiration from the human vision system and includes two horizontally displaced cameras in the same plane as shown in Figure 2.1.

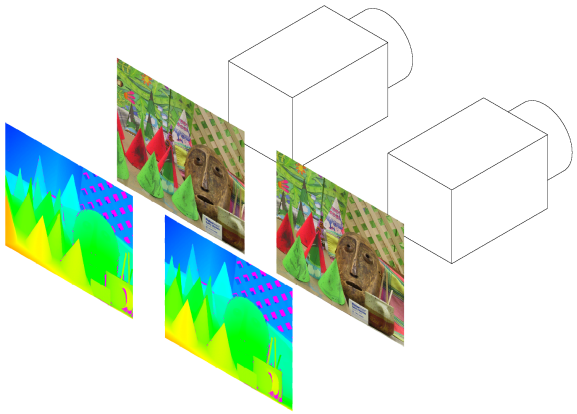


Figure 2.1: Binocular stereo vision process showing a typical camera rig, the left and right images as perceived by it, as well as two possible ideal colour-mapped depth representations of the input.

To illustrate how depth can be extracted from such a setup, said human vision system can serve as a practical example. Look at items at various distances while closing each eye alternately then note how objects will appear to jump left and right with moves inversely proportional to the distance to said feature [50]. By collecting these jumps, or *disparities*, for each point in the image we get the disparity space [49] of said image pair which we can refine into a proper depth map of the scene. While this simple test gives the impression of simplicity, actually finding the same match in the computer stereo vision case is a more complicated task and may not even be possible at all if the object is partially *occluded*, meaning it is only visible in one image.

In this section an introduction will be made to some stereo vision concepts, starting with camera requirements, then moving on to the actual matching step with a presentation of some common methods. Last a comparison with other ranging techniques is done.

2.1.1 Epipolar Geometry and Rectification

The practical experiment of Section 2.1 worked because the brain is able to correct for the relative placement of the eyes and their individual orientation. After this correction, object disparities in this example were limited left and right movement due to the horizontal alignment of the eyes. This camera correction process will need to be transferred to the case of two cameras before a proper matching step can begin. Without it the images would not be aligned, and the computer would have to search across the entire other image for each point it attempts to match, which would be a very time consuming process.

A point seen by a single camera only describes the projection of that point to the camera plane. The point itself can exist anywhere along a projected line in that direction as no depth is implied by just this point. The concept of *epipolar lines* describes how this projected line is viewed within the image plane of a different observing camera as shown in Figure 2.2. What this concept means is that when a computer lacking the human image context is to make a match, it no longer has to scan the entire other image looking for the best candidate, but can instead simply scan along this epipolar line. To find that particular line we must know the camera geometry or recover it from common points in the image [84]. A simplifying approach is to attach the cameras in the same plane with aligned horizontal lines. This mostly aligns the epipolar lines and forms the basis for the traditional horizontal stereo camera rig of Figure 2.1.

To increase the accuracy of the matching it is important to properly calibrate the rig by applying a transform to each camera and thereby aligning the image rows to each other projected to a common plane. This calibration, or *rectification*, can take many forms, but the resulting transform is typically solved by first photographing a known planar pattern at multiple angles then calculating the pose and distortion of each camera [83]. This process causes some image distortion along the edges which reduces the effective image size depending on how well aligned the cameras

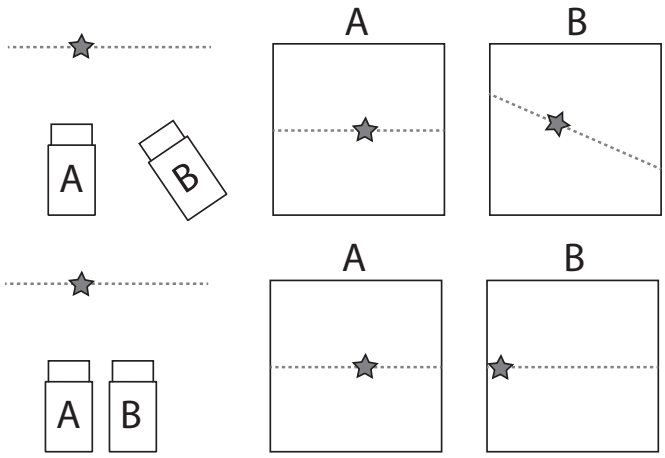


Figure 2.2: Searching for an object marked by the star in different camera geometries. The images as perceived by the cameras are on the right while the geometry is on the left. Note how matching can be done by only looking along the dotted lines and how the parallel binocular setup simplifies rectification.

were before rectification.

Before stereo calibration can occur it is important that slight *intrinsic* variation in the different cameras is compensated for as otherwise the same object may look inherently different in the two images making matching difficult. In this regard one must account for lens distortion and can expect sensor noise and exposure levels to vary even with sensors of the same make and model.

2.1.2 Stereo Correspondence

Stereo correspondence is matching points in one image with their corresponding point in the other within the region of camera overlap. This allows disparities to be calculated, and given the camera focal length and individual pixel size, the disparities can be related to depth via Equation (2.1). This gives a greater resolution at close range, as seen in Figure 2.3.

$$Depth(x,y) = \frac{baseline * focal}{pixelsize * disparity(x,y)} \tag{2.1}$$

The matching process may initially sound simple, but how exactly can we be sure that point A in one image is really point B in the other? This is known as the *correspondence problem* and is made more difficult by areas of low texture, repetitive patterns, brightness differences and reflective surfaces. An example may include two horizontally aligned cameras looking at a bright white wall through an image row aligned picket fence with a shadow covering one of the cameras causing uneven brightness in the output images. How can we now be certain of which plank of the fence should match which in the other image? And for a computer based approach, how can we match a single pixel on that plank when all other

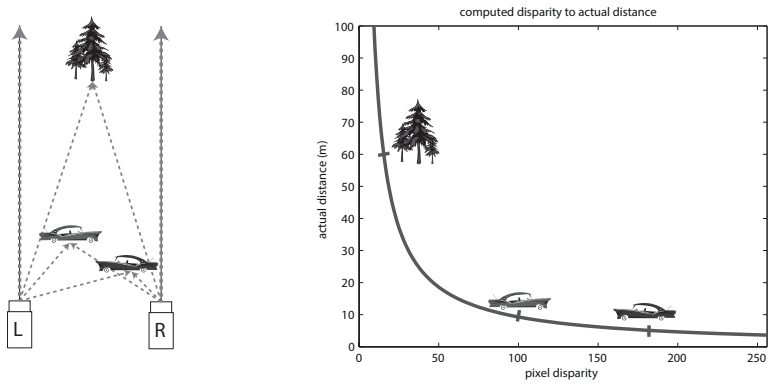


Figure 2.3: Example of the non-linear relationship between calculated disparity and actual depth. Exact relationship varies according to equation 2.1.

planks will have pixels matching it to some extent? In addition to this problem making a match should remove the matched point from future matches making the stereo correspondence problem NP-complete and in need of approximating solutions.

Two main paradigms are used to approach the stereo correspondence problem. The feature based sparse stereo methods match only the most certain points, while dense stereo methods use areas around each point to gather more information allowing more dense results to be had.

2.1.3 Sparse Stereo Correspondence

In computer stereo vision some points will be easier to match than others as they are easier to uniquely identify in the matching image. This opens the possibility of doing *sparse stereo correspondence* in which just the most robust features like edges, corners, or otherwise unique areas are matched between images. An example of this method can be seen in Figure 2.4. This leaves harder matches like low texture or ambiguous areas uncalculated, but region growing [24] and other surface fitting algorithms can be used to interpolate between these initial features. The advantages of the method lies in the high confidence answers on the feature points, even on imagery with differing illumination, as well as the limited computational effort required. Combined this led to early computer stereo vision work focusing on sparse correspondence methods.

Within the current field of stereo vision sparse correspondence with robust feature selection operators like SIFT [47], SURF [2], and FAST [64] remain useful for camera calibration [38, 83] and aiding dense stereo calculations [19]. Beyond these uses feature extraction has found use in related fields like finding road profiles, obstacles detection [45], face detection, mapping, and object recognition.

Figure 2.4: Example of sparse stereo correspondence using a feature extractor. Lines show the location of the point in both images. These locations can then be used to compute the disparity of that feature. Only the points with the most robust matches are kept, but what is construed as a good match can be tweaked to return more points at the cost of certainty of each new match. Made with OpenCV (see Section 2.4.2) using the FAST detector followed by a matching step. The Teddy image used is part of the Middlebury dataset [54].

2.1.4 Dense Stereo Correspondence

While sparse correspondence can find and identify points of interest, it is often useful for classification and tracking purposes to have a better segmented representation of the depth. The goal of *dense stereo correspondence* is then to calculate the disparity of every pixel within the reference image. This requires much more computational power than sparse correspondence and proves more challenging as ideally every area must get an assigned depth, even areas of partial occlusion (hidden to one camera), low contrast, poor lighting, and areas of repeating textures.

To simplify matching a few assumptions are made in regards to how the world is to be perceived. Surfaces are assumed to be *Lambertian*, this means that objects are assumed to have the same visual appearance in both images. In addition the world is assumed to consist of piecewise smooth surfaces [79]. These assumptions can be rewritten as an energy function $E = matchCost + smoothCost$. By minimizing this energy the goal of a stereo algorithm becomes finding the closest match while attempting to keep adjacent pixels at similar disparity. How the algorithm goes about this task is usually split into local and global methods.

As per Scharstein and Szeliski's work [66], dense stereo correspondence usually do a subset of the following steps:

1. Pixel-based matching cost computation on pre-rectified input. For a given pixel this determines a similarity measure to all its possible matches. Across all pixels this creates a disparity space image(DSI) [34] as can be seen in Figure 2.5. Common similarity measures include absolute and squared difference, but while simple to compute

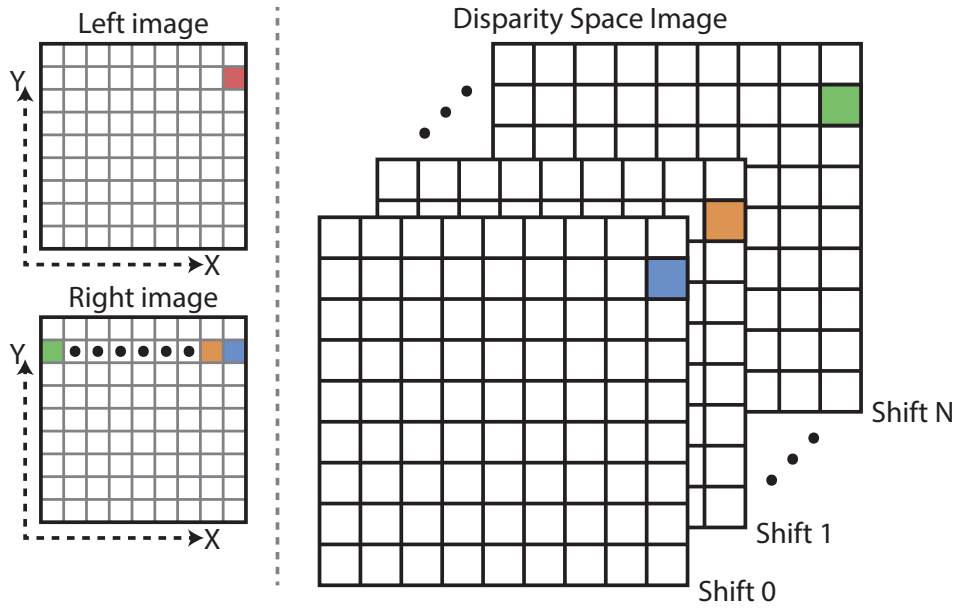


Figure 2.5: Disparity Space Image (DSI) showing how the pixel marked in the left image (red) is matched in the right image. In this example matching starts in the same spot in the right image and shifts leftwards while computing a matching score each step of the way. This matching score is stored in the DSI and used to compute the final best match.

they are easily affected by sampling and illumination variance. More robust measures include Birchfield et al. [3], normalized cross-correlation [24] and the census transform [80], though the latter two combine their work with the next step.

2. Cost aggregation. Deciding on depth using the direct results of the previous step tends to be influenced by noise in the data. To reduce this influence and provide smoothness as in the assumptions earlier, cost aggregation will usually be done across a support region. The simplest of support regions would be a fixed size square window wherein the matching cost is summed or averaged. Better windows are available and will be discussed in Section 2.1.4 Local Methods.

3. Disparity computation/search. This step is all about deciding which of the many possible matches in the previous step happens to be the best representative of the real depth. This can be as simple as a local winner takes all approach, or a more advanced global optimisation of the earlier mentioned energy function.

4. Post-processing / Disparity refinement. After the pixel correspondence calculation it is useful to further clean up the results. This is the subject of Section 2.1.5.

Global Methods

Global methods formulate the stereo matching problem as an explicit energy minimization problem wherein minimization algorithms are used to reach a global minimum error. Most of this work is then done in the disparity search step using an error value based on some variant of the earlier cost and smoothness equation. Current minimization techniques includes belief propagation [13], graph cuts[5] and dynamic programming.

These approaches have traditionally led to the most accurate stereo algorithms [54], but they are often too heavy on both computational resources and memory usage for real-time implementation [72]. This generally makes them poor choices for an evolutionary approach needing thousands of evaluations.

Local Methods

In contrast to the global methods, local methods do most of their work in the cost aggregation stage. A typical implementation will have a sliding support region looking for the closest pixel-wise match to the reference point. Common cost matching strategies include Sum of Absolute Differences (SAD) and Sum of Squared Differences (SSD), as described in Equations 2.2a and 2.2a. Here the pixel px has each of its potential disparites d computed by looking at the difference between its neighbourhood in the left image I_L and the neighbourhood of the potential match in the right image I_R. Other matching cost methods include more real-world photometrically robust variants like normalized cross-correlation [24], and the census transform [80].

$$SSD_{(px,d)} = \sum_{i \in window_{px}} [I_L(x_i) - I_R(x_i - d)]^2 \tag{2.2a}$$

$$SAD_{(px,d)} = \sum_{i \in window_{px}} |I_L(x_i) - I_R(x_i - d)| \tag{2.2b}$$

Through the use of windows a certain smoothness is implied as it averages out the noise of the image. The actual choice of window size can however be difficult as windows need to contain enough texture to ensure a proper match yet be so small as to only contain pixels of the same depth. This particular problem has seen much recent research including multiple windows [23], adaptive weights [31], and adaptive window sizes [81]. The latter two change their effective window size either through adjusting the importance of each pixel or by growing and shrinking the window dynamically. By only including similar areas in the effective support region this more or less solves the traditional window size problem.

For local methods the global energy-function is only implicitly optimised through locally optimal choices within the support region. By the use of this window function only a limited area of the DSI is needed to calculate each result which makes local methods faster, more memory efficient and with easier parallelization than global methods.

In the disparity search stage the easiest solution is the Winner Takes All (WTA) strategy. This merely assigns the closest match as the disparity. While very easy to implement this technique doesn't uphold uniqueness, that is a pixel in the non-reference image may actually end up mapped to several pixels in the reference image. This can lead to errors, but with a well-selected cost method results approaching the best global methods can still be achieved with the latest local algorithms.

A subset of the local algorithms are algorithms which while technically local still achieve behaviour similar to the global algorithms. Examples include cooperative algorithms [86] which are inspired by the human vision system wherein local computations are done iteratively, and hierarchical methods in which coarse initial results are used to guide increasingly detailed searches.

The most accurate local algorithms that still achieve real-time computation are currently the Semi-Global Matching (SGM) of Hirschmuller [25] and the ADCensus [51] algorithm which uses many of the same building blocks. SGM achieves great results by optimising each point in at least 8, but ideally 16 or more directions using Dynamic Programming. The original implementation uses a fair bit of memory, but a refined version which trades less memory usage for increased computing requirements, is available [26].

2.1.5 Post Processing

Post processing disparity images allows potentially bad correspondence matches to be removed or replaced by better choices. This step may include a *Left-Right disparity check* in which the computed disparity referenced to one image is sanity checked by seeing if the selected match would end up with the initial pixel if correspondence was done in reverse. This cleans up the disparity output and can be used to detect occluded areas [17]. Other common operations includes a subpixel interpolation step in which the neighbouring disparity candidates to the best match are fitted to a function, e.g. a parabola, and then used to assign a more accurate subpixel resolution match. Often such post processing steps are included within the actual matching when the relevant information is available, this allows more efficient implementations by reducing memory access and removing such results at an early stage.

Other operations may include filtering the output disparity map via a mean or median filter. If done with care this will reduce noise and improve the quality of the results. Results interpolation may also be used to fill in small gaps in the output image, but used in large areas this may give false impressions in tasks where it is more important to know whether a result has been calculated with a degree of accuracy over giving the best approximate estimate available.

2.1.6 Comparison to other ranging techniques

Stereo vision is an inexpensive technique able to exploit the latest in small high resolution cameras. The sensors are passive and low power, offers no moving parts, high definition of the results, and with potentially high refresh rate. In addition they offer the additional option of using the same images for scene context, allowing e.g. signs, turn signals and brake lights to be analysed using the same hardware.

The main problems remain light variation, use in poor visibility, hardware requirements, and temporary solution degradation [33]. The following sections will provide a short introduction to sensors often used in parallel to, or instead of a camera rig.

Lidar

In autonomous vehicles an oft used method is the lidar, a laser based technique in which one or more beams of light are projected to the scene while a detector measures the time till the signal is reflected and returned. A typical 2D lidar configuration features one beam mechanically swept to get readings along that particular plane. More advanced, and expensive, 3D lidar models feature a column of beams all attached at slightly different angles to cover a larger vertical portion of the scene as the device is rotated. While rotating each return is recorded and optionally corrected for device movement before a point cloud is generated as in Figure 2.6. In this case the horizontal resolution is limited by the speed and accuracy of rotation as well as the bandwidth required for handling the number of points generated.

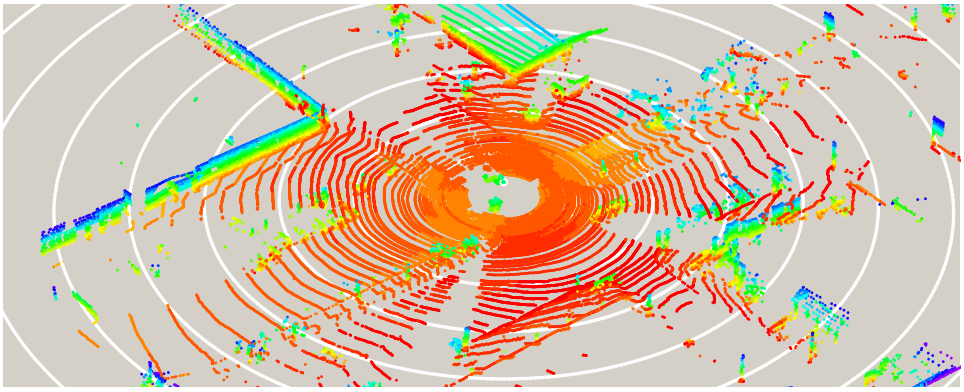


Figure 2.6: Lidar point cloud as generated from a single rotation of a Velodyne HDL-64E lidar unit mounted on a vehicle standing at an intersection. Points are colour-mapped based on the height of the obstacle from a low red to a tall purple. White contour lines are 10 meters apart. Image generated based on data from the Ford campus vision and lidar data set [60].

This approach provides excellent accuracy and ability to detect the presence of objects, but the limited vertical resolution and range-dependent horizontal resolution may make actual identification of objects difficult. For

the decision-making process different decisions may have to be made based on whether an object is a post or a pedestrian, as different behaviours are assumed. This is usually solved with a sensor fusion approach in which the lidar data is either guiding or otherwise compared to camera data [63].

Lidar generally provides robust range finding and is unaffected by time of day as well as bright sunlight and otherwise varying lighting conditions. However as with a camera based range sensor lidar is affected by rain and snow, and may also report dust, fog and exhaust as a collidable objects [61, 76].

Radar

Radar is a widely used sensor, particularly in the automotive setting, were its seen use in cruise control [76], automatic collision avoidance, lane assistance, and parking assistance applications. A radio wave is transmitted and the return signal timed and analysed, allowing the range and target speed to be interpreted. The latter is through the Doppler effect and provides context to detected obstacles. If driving and an obstacle is in front of you it is rarely a problem if that car is moving the same direction.

Radar devices tend to be either fixed, which is very common with parking assistance sensors, or mechanically swept. A more recent development in the automotive field is the phased array radar which uses digital beam forming allowing the radar array to stay stationary yet still sweep a beam within a certain field of view. The primary benefits is the lack of moving parts and increased resolution. An interesting additional feature however is the ability to bounce a beam off the road surface allowing a look past the car ahead of the radar equipped vehicle.

While radars are weather independent they can still report spurious results. E.g. a car passing on a bridge far ahead, or in a different lane, may be reported as an actual obstacle. The resolution as such is fairly poor.

2.2 Stereo Algorithms

This section first explains how stereo algorithms can be rated and common evaluation data sets used. Lastly stereo algorithms used in this thesis are presented in some detail.

2.2.1 Evaluation of Algorithms

Stereo algorithms are typically evaluated by comparing the resulting disparity image with a *ground truth* (GT) image. The ground truth image is a representation of the real underlying disparities as seen in the scene from one of the two camera viewpoints. In the case of computer generated test images this ground truth image can easily and accurately be created from the model of the scene. Computer generated images provide a controlled method for testing specific aspects of an algorithm, but may not represent all the nuances of a real-world stereo pair [75]. In contrast test images of lab

and especially outdoor scenes work well as general algorithm evaluators, but providing the ground truth of the visible scene requires a different approach as a 3D model and perfect knowledge of ones own position is rarely available.

One way to generate such a ground truth image is through the process known as structured light in which a series of patterns are projected to the scene while a camera captures how the pattern is distorted by visible objects [67, 68]. This creates highly accurate images, however, as the basic structured light technique relies on interpreting visible light from a projector, its accuracy is reduced both by distance and by ambient light diffusing the pattern. The number of patterns required for the best quality results and the multiple exposures of each also precludes scenes with movement. These limitations makes the method very well suited to lab and other static environments, but less suited for outdoor scenes.

For outdoor scenes a laser scanner approach is typically used in the form of a LIDAR (see Section 2.1.6). This produces a sparse point cloud of the scene which can then be translated to the viewpoint of the camera vision system and compared to the stereo correspondence results.

Quality Metrics

A number of metrics are used for comparing the output of the stereo algorithms with the ground truth. These measures typically compare the GT image and the result on a pixel by pixel basis. however, due to consistency checks and problematic regions even normally dense stereo algorithms (see Section 2.1.4) may produce areas with no set disparity level. Whether such pixels have been rejected or are simply uncalculated, they are nonetheless marked as invalid as per Equation (2.3b). Any calculated pixel may contribute to a number of error measures like the remaining Equations in 2.3. These are widely used in published results and benchmarks (see Section 2.2.1).

$$E_{Bad} = \frac{\sum_{y=0}^{height}\sum_{x=0}^{width}(|d(x,y) - d_{GT}(x,y)| > Bad_{Threshold})}{N} \tag{2.3a}$$

$$E_{Invalid} = \frac{\sum_{y=0}^{height}\sum_{x=0}^{width}(d(x,y) == unmarked)}{N} \tag{2.3b}$$

$$E_{Out} = E_{Bad} + E_{Invalid} \tag{2.3c}$$

$$E_{AvgBad} = \frac{\sum_{y=0}^{height}\sum_{x=0}^{width}|d(x,y) - d_{GT}(x,y)|}{N_{marked}} \tag{2.3d}$$

$$E_{RMS} = \sqrt{\frac{\sum_{y=0}^{height}\sum_{x=0}^{width}|d(x,y) - d_{GT}(x,y)|^2}{N_{marked}}} \tag{2.3e}$$

These metrics may be applied to the image as a whole or only in an area of interest. Applying a mask is useful for studying a particular quality in potentially problematic areas areas like partially occluded regions, low

texture, or reflective areas. This in turn may give a better understanding of the particular strengths and weaknesses of each algorithm or indeed the parameters controlling it.

In addition to evaluating the resulting image, an important concern remains at what pace the result was calculated. This metric is more difficult to compare as the resulting time will not only depend heavily on the processor architecture and memory of that particular computer, but also on the actual implementation of the algorithm in question. As such comparisons can only be directly made within the same setup. Even so, runtime remains an important concern for real-time implementations and comparing the efficiency of algorithms. While it would be possible to compare the times spent directly, it is equally important to know how well an algorithm copes in regards to image size and the number of desired disparity levels. Such measures are included in the oft used measures of *seconds per megapixel* (s/MP) and the *million disparity levels per second* (Mde/s) metric as per equations 2.4.

$$s/MP = time * \frac{1000000}{width * height} \tag{2.4a}$$

$$Mde/s = \frac{width * height * disparities}{time} * \frac{1}{1000000} \tag{2.4b}$$

Stereo Datasets and Benchmarks

Good test data is required to get a handle on the performance of a stereo correspondence algorithm. Several datasets are currently publicly available, some of which are presented here.

The Middlebury dataset [54] originally created for the stereo taxonomy of Scharstein and Szeliski [66] has grown to become the de facto standard for testing stereo algorithms. It provides a series of stereo pair images as well as ground truth solutions describing the ideal disparity map for each part of the dataset. Algorithms are ranked based on their performance in normal non-occluded areas, performance near depth discontinuities and occluded areas. The results are publicly available along with links to the scientific papers of each algorithm.

While the Middlebury set provides an excellent driving force for research it has been shown that the lab ranking may not always match real-world performance [46]. This has lead to an increased focus on real-world problematic scenes with the desire for more robust results. The Middlebury benchmark is currently experimenting with a new updated dataset which includes such test scenarios.

The KITTI dataset [20] provides a number of real-world stereo images and sequences as recorded from a moving vehicle in Karlsruhe, Germany. The set is based on data from both greyscale and colour cameras, and is backed up by lidar provided ground truth as well as GPS and accelerometer data. When compared to the Middlebury set this real-world focus presents more of the potential problems a stereo algorithm may face in regards to noise and lighting conditions, which in turn should lead to more robust algorithms [18]. The dataset lays the foundations for the KITTI benchmark

and has found use in various autonomous vehicle related fields including stereo vision, optical flow and object recognition.

Beyond the datasets an important consideration remains the platform the implementation is to be used. In the case of a real-time implementation it is often required to simplify the algorithm to achieve the required throughput. Another concern may be how easily the algorithm can be parallelized for use in parallel hardware like graphics cards and programmable logic devices. Surveys on the applicability of algorithms in the real-time setting includes the work of Tippets et al. [72] and the more vehicle-oriented work of Mroz and Breckon [56], and Van der Mark and Gavrila [74].

2.2.2 OpenCV Block Matching (BM)

Block Matching as implemented in the OpenCV library (see Section 2.4.2) is a very fast stereo implementation based on the Small Vision System algorithm of Konolige [43]. BM is a purely local stereo correspondence algorithm which finds initial matches via small Sum of Absolute Difference (SAD) windows. As per Scharstein et al. [66] BM can be divided into initial matching, disparity optimisation and disparity refinement, with additional preprocessing of the input images.

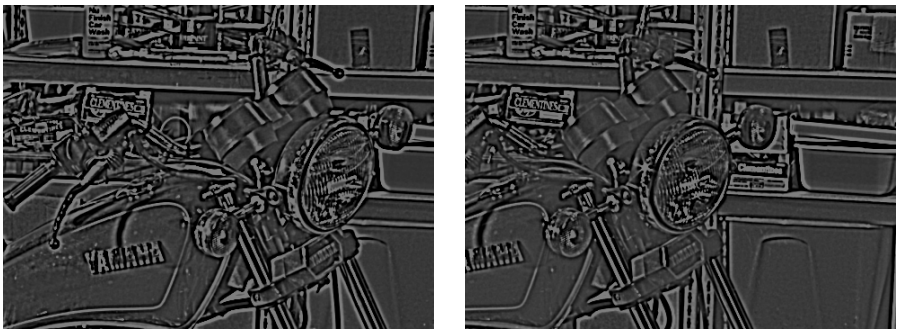
Typically local window methods are sensitive to brightness differences in the image pair. The preprocessing step of BM includes image intensity normalization wherein the brightness of the images are adjusted by the mean intensity within local windows. An example of the output of this normalization approach can be seen in Figure 2.7b. Note how the box in the bottom right now has the same apparent intensity in both images. Also of note is the reflective surfaces in the image and how they still differ in appearance, making matching difficult.

The alternate and preferred preprocessing method of BM is the XSobel edge extractor (see Figure 2.7c). The goal of this prefilter is to extract more robust features for the matching step, and as edges tend towards stability within the image pair they make for a good choice. Matches are done horizontally, hence the Sobel filter is applied only in the X direction thereby highlighting edges crossing the epipolar lines. This implementation differs slightly from the original Konolige algorithm as originally a Laplacian on Gaussian (LoG) filter was used for combined smoothing and edge extraction.

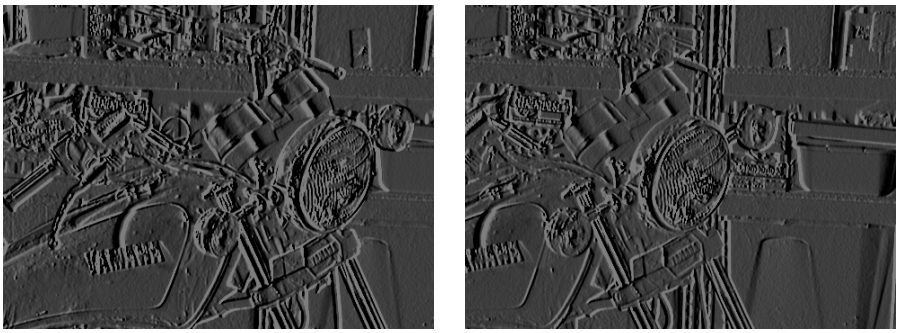
The initial matching of the block matching algorithm is done using Sum of Absolute Difference (Equation (2.2b)) using a sliding window technique. Matches are however only assigned when there exists an acceptable amount of texture along the epipolar line direction. Additionally, for a match to be accepted it has to uphold a *uniqueness constraint* by being a certain ratio better than the second best match. Should this be in order a match is assigned and the disparity is calculated based on an interpolation between the neighbouring values giving 16 subpixel disparity resolution.

As a window based method the initial matching may have problems near object discontinuities as the window will contain parts of both the

(a) The left and right input images



(b) The left and right image as adjusted by the normalized response prefilter



(c) The left and right image as output from the XSobel prefilter

Figure 2.7: The output of the normalized response filter (b) as implemented in the Block Matching algorithm. XSobel prefilter (c) as implemented in both BM and the Semi-Global Block Matching algorithms. Input images are cropped versions of the MotorcycleE image pair of the Middlebury benchmark [54]. This image pair features unequal brightness for robust testing. (prefilter size 9, prefilter cap 63)

background and the object. The background will also change from one image to the next due to perspective differences, which means that this issue may lead to potentially poor matches. BM compensates by carrying out a speckle removal step in which results within areas of large variation are removed. A left-right disparity check is also applied to further remove spurious matches.

2.2.3 OpenCV Semi Global Block Matching (SGBM)

The OpenCV Semi Global Block Matching algorithm is a blend of the BM algorithm (see Section 2.2.2) and Hirschmuller's Semi Global Matching (SGM) [25]. Like the BM algorithm SGBM matches windows centred at each pixel with a uniqueness threshold applied. However, rather than the SAD metric a more accurate Birchfield Tomasi [3] cost metric is used. The XSobel prefiltering used by BM is incorporated into this matching step.

As with BM a uniqueness constraint is used to assist in good matches, but unlike BM a dynamic programming approach adapted from the SGM algorithm is used to improve the final matching. This variant is more limited than the one used in SGM and by default the dynamic programming is only used along rows to optimise the best disparity choice at each pixel. Inter-row consistency is not upheld by dynamic programming, but rather by a greedy approach. This mode, called the 5-way dynamic programming mode, allows the result to be computed in a single pass thereby greatly increasing the throughput over the SGM algorithm. An alternative slower two-pass version using 8-way dynamic programming is selectable through a control parameter. As with BM a subpixel interpolation step is done to increase the resolution of the resulting disparity.

The left-right check is handled within the disparity selection, and a median blur is applied to the end result. Lastly a speckle filtering step, as in BM, helps improve the accuracy of the result.

2.2.4 ELAS - Efficient LArge-scale Stereo

The Efficient LArge-scale Stereo (ELAS) algorithm by Geiger et al. [19] is a local method guided by robustly matched support points taking advantage of how some points are more easily matched than others. ELAS first applies a sparse grid across the image, then for each intersection a potential support point check is carried out. A point is a potential support point if the best match found in the other image is at least a factor τ better than the next best match. Before such potential support points can become actual support points for the next step in the algorithm, the following refinement steps are done:

1. A left/right-consistency check must be successfully carried out within a threshold.

2. Potential support points which disagree with their immediate neighbours are removed as inconsistent points.

3. When multiple points lie on a straight line, only the ends are kept as the points are deemed redundant and would interfere with the triangulation of the next step.

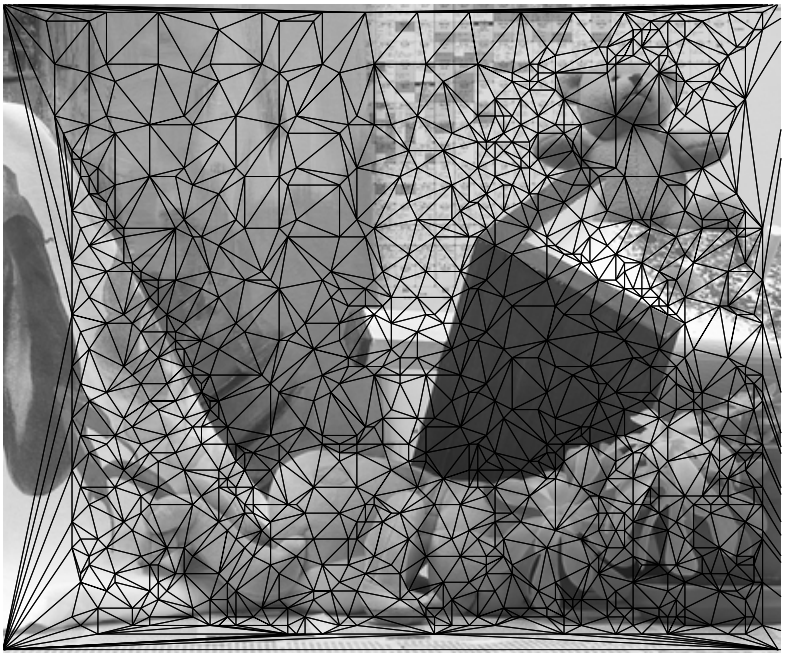


Figure 2.8: An example of the Delaunay triangulated support points of the ELAS algorithm using its Middlebury default parameters. The background image is the Teddy image of the Middlebury benchmark [54].

With the support points created the next step of the algorithm is to create a mesh based on these points so that the local search can be guided in an efficient manner. An intermediate flat mesh based on the support point image coordinates alone, is created via Delaunay triangulation. The Delaunay process defines triangles between all points while maximizing the minimum angle of each triangle. Figure 2.8 shows an example output from this process. Once this mesh has been created support planes are created between the support points with their Z-axis defined by their calculated disparities.

The generated planes are used in the matching step to greatly reduce the disparity search space based on the assumption of piecewise smooth disparities. By combining multiple planes more certain matches are generated. As with the support points a left-right disparity check is also used for more robust matching.

Built in post-processing includes speckle removal, result interpolation, and bilateral mean and median disparity filtering.

2.3 Evolutionary Algorithms

Evolutionary algorithms is a class of generate and test optimisation algorithms inspired by the biological process of evolution. The basic

premise is a population of candidate solutions to a problem, tested and rated on data relevant to the optimisation. Then a survivor of the fittest procedure, in the form of a *survivor selection* step wherein good solutions are kept and bad ones potentially discarded, is carried out. The selection of the remaining candidates are combined to form new offspring and the process repeats giving a steadily increasing total population score, or *fitness*.

Unlike an exact solution an evolutionary approach does not guarantee that the global best solution to a problem is found. However, with good methods and testing procedures, good solutions can be found even on unknown problems without having to negotiate the entire *search space* of candidate solutions.

2.3.1 An Introduction to Genetic Algorithms

This section will focus on the Genetic Algorithm (GA) as described by Holland [29, 30]. Genetic algorithms are a popular subset of Evolutionary Algorithms and likewise work in an iterative process on a population of candidate solutions. The basic outline of such an approach is illustrated in Figure 2.9.

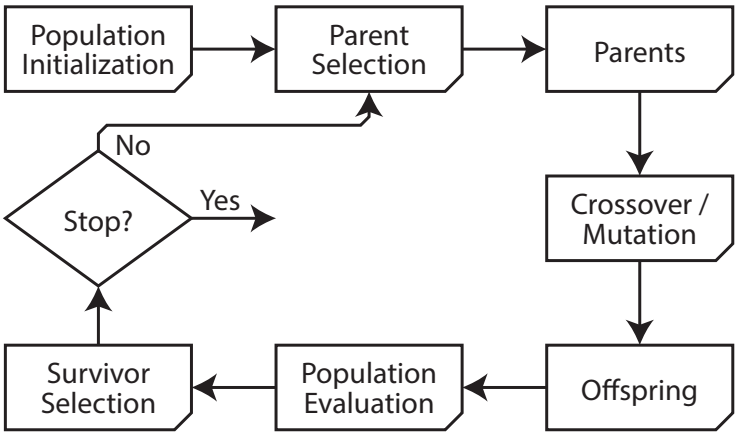


Figure 2.9: An overview of the flow within a typical genetic algorithm.

Before the process can start the problem to be optimised will need to be represented in a way compatible with the GA and its operators. This takes the form of mapping problem specific parameters into a gene format representing the *genotype* for each individual. Problem parameters may be represented using a traditional binary representation, integer, or real-value representations among others. Running this genotype through the problem at hand creates the *phenotype* representation for that individual and can be used to *evaluate* its performance. Evaluating the performance is done by a *fitness function* which assigns a score based on how well the individual performed the task represented by the problem.

Once a problem representation has been chosen the genetic algorithm

can be started by creating a random initial population or by inputting values of interest to guide the search to that region of the search space. With this done the population is evaluated by the fitness function and each individual is assigned a *fitness* score. The GA then proceeds to its main program loop starting with parent selection, which will be the subject of the next section.

Parent Selection

At the start of each generation parent selection is applied to select the best individuals for creation of new offspring. The traditional approach is to select two parents, but multiple can also be selected [12]. Many methods exist, but most of them prioritize high fitness parents, while a random element ensures that a diverse set of parents are selected and thereby giving the best possibility of advancing the search through new offspring created through the genetic diversity operators of *crossover* and *mutation*.

Example selection operators include *fitness proportional selection* wherein each candidate solution is assigned a portion on a roulette wheel proportional to its fitness, then the dice is cast and parents selected. The popular *tournament selection* works by selecting t random individuals then making a parent of the one with best fitness. The basic version uses a tournament of size 2, but larger tournaments may be used if a greater degree of selection pressure is desired. This process continues until the requisite parents have been found. This method implicitly ranks the population without requiring knowledge of the current fitness values of the entire population [4].

The Crossover Operator

Crossover is a genetic operator wherein two or more parents create one or more offspring by combining their genetic material according to specific crossover rules. The goal of this recombination process is to combine good parents in the hope of creating new even better offspring exploring new areas of the search space. Whether the operator is applied is controlled by a crossover rate parameter giving the probability that new offspring is generated or the parents themselves kept directly. The latter describes stationary crossover while the former will either become a contracting or expanding crossover depending on the parameters. Contracting and expanding refers to whether the offspring gene is generated between the corresponding parent genes or outside of them respectively.

Among the simplest forms of this operator is the *one-point crossover*. Given two parent genomes it generates two offspring by first selecting a random point within the genome, called the *crossover point*. Genes from the first parent before this point are assigned to the first offspring, while genes past this crossover point is assigned to the second. The genes of the second parent is likewise distributed first to the second offspring, then the latter gene portion to the first offspring. This operation can be extended to a *two-point crossover* type by selecting two random points to split the genome. This is illustrated in Figure 2.10. Both these crossover types work

with both real-valued and binary genomes, and neither of them alter the gene average. However they exhibit a certain *positional bias* in that the chance of including a gene is dependent of the position of that gene.

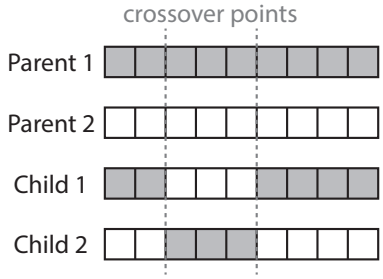


Figure 2.10: The workings of the two-point crossover operator with two example crossover points.

The *Simulated Binary Crossover* (SBX) [8] operator is a crossover variant which attempts to simulate the behaviour of the binary single-point crossover on real-valued genotypes. As with the single-point crossover this preserves the average parent gene value. Offspring is created based on a distribution model with the tendency to create new offspring close to the parent values with less variation the closer the parents are to each other in that gene. The distance from the parents is controlled by a crossover distribution value η_c and a randomly drawn number $u \in [0,1]$. For two parents p_1 and p_2, two children c_1 and c_2 are created with their distribution controlled by Equation (2.5) and their individual genes i assigned via Equation (2.6).

$$\beta_q = \begin{cases} 2u^{\frac{1}{\eta_c+1}} & \text{if } u <= 0.5 \quad \text{(Contracting)} \\ \frac{1}{2(1-u)}^{\frac{1}{\eta_c+1}} & \text{if } u > 0.5 \quad \text{(Expanding)} \end{cases} \tag{2.5}$$

$$c_{(1,i)} = 0.5((1+B_q)p_{(1,i)} + (1-B_q)p_{(2,i)}) \tag{2.6a}$$
$$c_{(2,i)} = 0.5((1-B_q)p_{(1,i)} + (1+B_q)p_{(2,i)}) \tag{2.6b}$$

The Mutation Operator

The mutation operator introduces new genetic material by randomly altering one or more genes of the genome of a single individual. The exact implementation varies with the encoding of the genotype with many different kinds of mutation operators. A simple *bit mutation* may be applied to a binary genome wherein a random number is generated for each bit, and if it is within a certain *mutation rate* threshold, the bit is reversed.

In *polynomial mutation* [9] a real-valued gene is altered slightly by a polynomial distribution. First a random $u \in [0,1]$ is generated and the assigned distribution is picked as in Equation (2.7a). The mutation distribution index η_m controls the shape of the distribution. A child gene

is then updated based on Equation (2.7b) where Δ_{max} sets the maximum allowed change in the gene.

$$\bar{\delta} = \begin{cases} (2u)^{\frac{1}{\eta m+1}} - 1 & \text{if } u < 0.5 \\ 1 - (2(1-u))^{\frac{1}{\eta m+1}} & \text{if } u >= 0.5 \end{cases} \tag{2.7a}$$

$$c_i = p_i + \bar{\delta}\Delta_{max} \tag{2.7b}$$

Survivor Selection

To keep the population size constant the total offspring and parent populations will have to be reduced so that it fits within the population size constraint before the start of a new iteration. This takes the form of a survivor selection step usually based on individual fitness wherein survivors with higher fitness are given priority. This may be accomplished with a randomized selection with many of the same methods used in the parent selection routines. Selection may also be controlled by attempting to uphold a certain diversity, or even a simple age-based routine in which newer individuals take precedence over older ones.

In this stage it is important to consider the possibility of dropping the best found solutions and ways to avoid it. The *elitism* mechanic deterministically selects the very best fitness individuals and lets them propagate to the next generation. This may represent just one, a few individuals, or the entire new population may be filled directly by fitness only. The latter leads to a rapidly converging search, but may suffer from loss of genetic diversity and converge to a local minimum.

2.3.2 Evolutionary Multiobjective optimisation

In a single-objective optimisation problem a global optimum may be found which is better than all other solutions for that given problem. However many practical problems have multiple conflicting objectives like cost, time, safety and performance, which may not all simultanously reach their optima. In these cases the notion of a single best solution no longer holds as a compromise would have to be picked. It is possible to reduce some multi-objective cases to single-objective problems by combining objectives as a weighted sum [35] through a process known as *scalarization*. However this requires that sufficient knowledge of the scale of the fitness landscape exists, such that a desired trade-off amongst objectives can be chosen in advance with appropriate weights set. This prior knowledge may not be available and even so, small changes in the weights of the fitness function can lead to large changes in the direction of the search and the final best solution found [15, 71].

Evolutionary Multi-objective optimisation (EMO) algorithms change this approach from looking for a single best solution to searching for many great trade-offs all at once. Work in this field started in earnest with the early work of Schaffer and his VEGA [65] algorithm. VEGA treated the multi-objective case by dividing the selection step into slots dedicated to

each objective wherein parents were selected based on that objective alone. After a shuffling step the selected candidates were crossed as in a normal genetic algorithm and new offspring created. This approach tended to bias the solutions to extremes of each objective making good compromise solutions difficult to find. Additionally while the author mentions an external population for continuously saving good solutions, this is not implemented, and so, without elitism, the algorithm may drop good trade-off solutions between generations whenever that solution is not near the best of any objective. This and subsequent algorithms of the same era illustrated the problems associated with achieving enough, diversified, and good compromises to solve the multi-objective case.

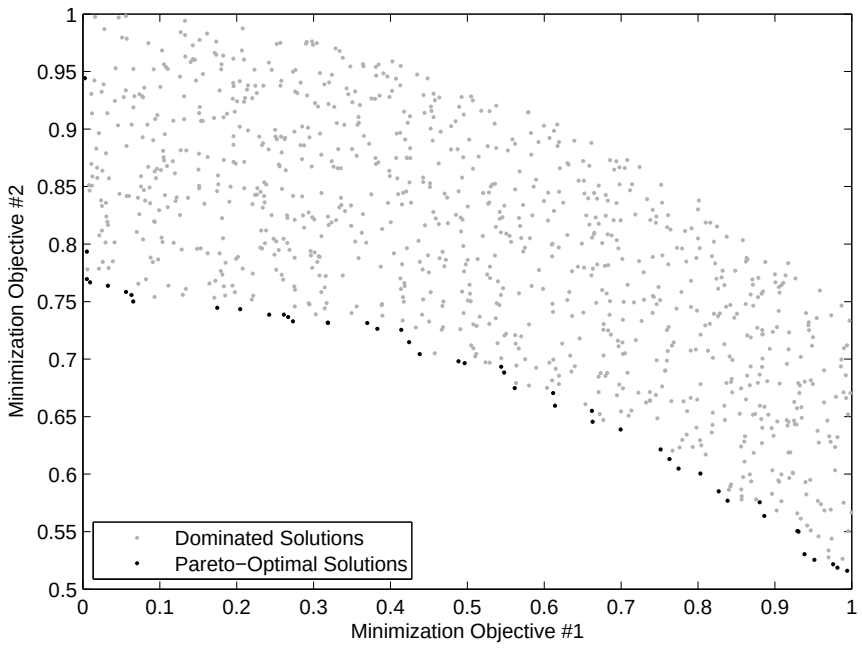


Figure 2.11: An example of pareto-optimal solutions in a two-objective minimization problem.

Current EMO algorithms correct this approach in various ways usually taking the form of applying a niching procedure to spread solutions in the search space while taking into account all objectives. When selecting and comparing solutions this then requires a new way to compare solutions and finding which are strictly better than the rest. A common way is to employ the concept of *pareto-optimal solutions*, which represent all so-called *non-dominated solutions* for that problem. A solution is said to dominate another if it is better in at least one objective while remaining no worse in all other objectives. In the end solutions not tagged as dominated are indicated as superior choices and form what is known as the Pareto-optimal front, as can be seen in Figure 2.11. In other words the solutions in the Pareto-optimal set are solutions which cannot be improved in one objective without simultaneously deteriorating in another. The solutions within the pareto-optimal set can't be said to beat each other overall, but

given a desired trade-off between objectives, one of them will be the best choice for the problem at hand. Presented with the pareto front it is then up to the user to select the desired trade-off, a process that may now be easier and better informed given the greater insight afforded by the potential solutions.

Popular EMO algorithms include Strength Pareto Evolutionary Algorithm 2 (SPEA2) [88], Pareto-Archived Evolution Strategy (PAES) [40], MOEA/D [82], and Non-dominated Sorting Genetic Algorithm II (NSGA-II) [11] among others.

The Hypervolume

When comparing the pareto front outputs of multiple algorithms it may be difficult to objectively determine the best result. One quality metric capable of reducing the multi-point pareto front to a single quality indicator is the hypervolume measure as described by Zitzler et al. [87]. The hypervolume describes the volume covered in the objective space by the solutions of a given pareto front. In the base 2-objective case it can be computed by taking the union of rectangles formed between each pareto-optimal point and a given upper objective bound, as shown in Figure 2.12. Notice how each pareto-optimal point has its own exclusive contribution to the final area. This 2D base case can then be extended to multiple dimensions to handle an increase in objective counts.

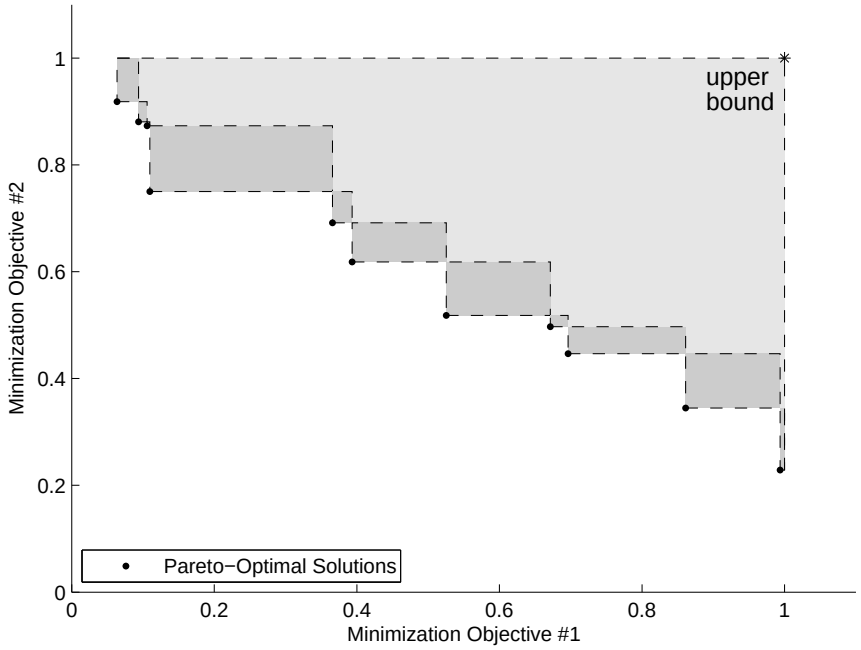


Figure 2.12: The hypervolume - a quality measure for pareto fronts. Shown with two normalized minimization objectives and a (1.0, 1.0) upper bound. The shaded areas describe the hypervolume. The darker shades represent the exclusive contribution of the adjoining pareto-optimal point.

Beyond comparing algorithms after the fact, the hypervolume can also

be used as an objective in the selection function of common evolutionary approaches. Selection based on maximizing the hypervolume contributions of each individual enforces population diversity and leads to a well-defined front [14]. However, given its computational complexity which is exponential in the number of dimensions, an approximating or otherwise simplified solution may be used in practice.

Current research

Research in the evolutionary multi-objective field is currently focused on problems with more than 3 objectives as this provides a challenging landscape for current algorithms. As the number of objectives grows the fitness landscape increases in dimensionality and the required points to define a pareto front increases to a point which few solutions will be outside the pareto-optimal approximation. This leads to poor convergence. In addition the niching step of current algorithms tends to become computationally expensive as the number of objectives grows [10].

If the current solutions are clustered far apart then recombination without taking the clusters into account may lead to slow progress as the offspring is unlikely to be of use in better defining the current front.

2.3.3 NSGA-II - Non-dominating Sorting Genetic Algorithm

The Non-dominating Sorting Genetic Algorithm II (NSGA-II) of Deb et al. [11] is an evolutionary multi-objective algorithm based around the concept of dominance and ranking. It is an improved version of the original NSGA, a popular and at the time innovative solution to the multi-objective problem [71]. Changes over the original version includes the addition of elitism to prevent the loss of good solutions, normalized crowding distance as parameter independent diversity operator, and a faster non-dominated sorting approach.

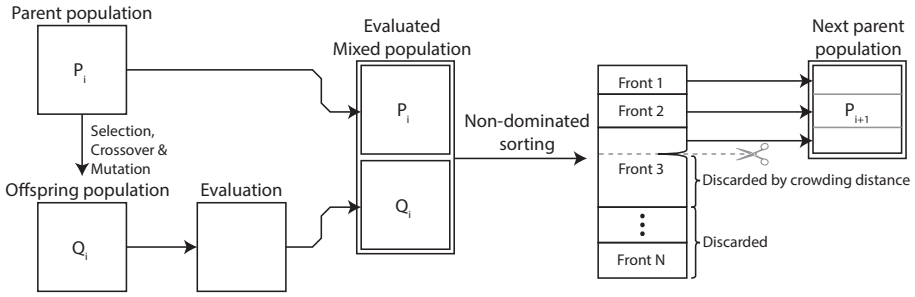


Figure 2.13: The basic flow of NSGA2.

The NSGA-II process is illustrated in Figure 2.13. It starts with a random initialization of what it refers to as the parent population, but note that this parent population does not illustrate the actually selected parents,

as in the basic GA, merely the current population. This population is then sorted using non-dominated sorting wherein the individuals on the pareto front are assigned rank 1, then if these individuals were to be removed the next front is assigned rank 2 and so on. The actual implementation uses a faster counting scheme giving a complexity of $\mathcal{O}(mN^2)$, where m is the number of objectives and N is the population size. Their rank becomes the main fitness of each candidate solution. The next step is to create offspring based on tournament selection, recombination and mutation. The selection is based first on rank, then crowding distance, and lastly a random draw if not resolved. After evaluation this offspring population is combined with the parent population and the ranking procedure is repeated. As this population is too large, survivor selection cuts it down to the designated size by first filling the new population based on rank, then if a rank is unable to fit in its entirety the crowding distance within that rank is used to assign survivors thereby keeping diversity. The *crowding distance* is the sum of the difference in objectives to the neighbours of each solution, with a special guard value assigned to individuals at the edge of each objective, giving them precedence over others during survivor selection.

A recent development of this algorithm is the new NSGA-III [10] variant. This is focuses on problems with more than 3 objectives, tackling the issues this increase in search space entails. It does this by changing the crowding distance to account for the closeness to reference points, thereby adding guides to the search.

2.4 Tools

This section presents some the statistical tools and software used while working on this thesis. Table 2.1 summarises the software used.

Purpose	Name	Version
Images	Adobe Photoshop	13.0.1
Illustrations	Adobe Illustrator	16.0.0
Statistics and graphs	Mathworks Matlab	2013b & 2014b
Text editor	Sublime Text	2
C++ development	Microsoft Visual Studio	2013 & 2014
Computer Vision Library	OpenCV	2.4.9, 3.0 alpha, & custom 3.0
Parallel Computing Library	Intel TBB	4.3
Stereo Evaluation	Middlebury SDK	1.3
Stereo Evaluation	KITTI SDK	-
Disparity Visualization	Computer Vision Toolkit (cvkit)	1.5.0

Table 2.1: Summary of most of the software used during work with this thesis.

2.4.1 Statistics

When evaluating evolutionary performance it is useful to be able to both illustrate the results and analyse whether one result can be shown to be

statistically better than another. This section will give a brief overview of some of these methods used in this thesis.

Boxplot

Boxplots are a useful tool for visualizing results allowing a better understanding of their distribution of values beyond what simply plotting the mean or median would do. Figure 2.14 illustrates a boxplot using the Tukey variation for whisker values [16]. First the median value of the items to be plotted is calculated, giving the 50th percentile (2nd quartile) marked in the plot. Splitting the population at the median gives two new groups of values each providing their own median at the 25th and 75th percentiles (1st and 3rd quartile) as in the illustration. The box itself represents the values lying within these latter two quartiles and defines the Inter Quartile Range (IQR). Whiskers are in turn defined as the last value within 1.5*IQR of the outer quartile values, which is roughly equivalent to covering 99.3% of all plotted values. Any values left are plotted as value outliers.

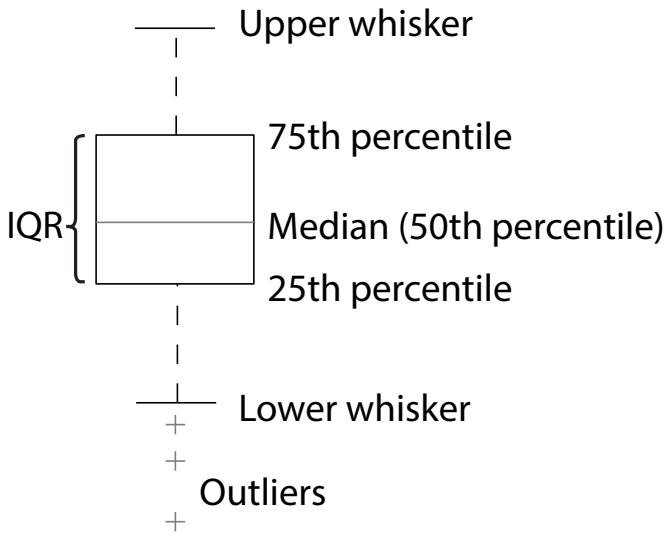


Figure 2.14: An illustration a boxplot with labeled features.

The Wilcoxon Rank-sum test

The Wilcoxon Rank-sum test [77] is a statistical test used for comparing two populations of values without prior knowledge of their underlying distributions. By not making this assumption it falls into the class of non-parametric tests. This is done by creating a U-value indicating whether the *null hypotesis* of equal population medians is true. The rank-sum method is equivalent to the Mann-Whitney U test and works by first sorting the two populations, then the minimum value among the two is extracted and assigned rank 1. This operation continues with ranks assigned in ascending order, with tied values assigned the mean of the ranks they otherwise would have received. After all values have been ranked the first population ranks

are tallied giving its sum of ranks R_1. The population U-value can then be calculated using Equation (2.8), where N and M are the first and second population sizes respectively.

$$U_1 = N * M + \frac{N * (N+1)}{2} - R_1 \tag{2.8a}$$

$$U_2 = N * M + \frac{M * (M+1)}{2} - R_2 \tag{2.8b}$$

The smallest of these U values gives an indicator of the closeness of the two value populations with a U of 0 being evidence of very dissimilar value distributions. This metric can then be converted to a p-value which illustrates the statistical significance of the difference. Typically, for statistical robustness a p-value of less than 0.05 or 0.01 is used for 95 and 99 percent significance respectively.

2.4.2 The OpenCV library

The Open Source Computer Vision library [6, 59] is collection of algorithms aimed at making the field of computer vision more accessible to both programmers and researchers. OpenCV started as a project at Intel Research Initiative in 1998 and has since been transferred to the non-profit OpenCV.org foundation currently supporting the library. The library is freely available under an open source BSD 3-Clause license[1], and supports multiple platforms including Windows, Linux, Mac, iOS and Android. OpenCV features more than 2500 algorithm implementations including image filters, image stitching, machine learning, object detection, and optical flow among others. As many of the basic algorithms are transferable to other fields this wide range of implementations has allowed OpenCV to be used outside of the originally intended field of computer vision.

For software stereo vision in particular the library implements a block matching algorithm derivate of Konolige [43] as well as a block variant of Hirschmuller's semi-global matching (SGBM) [25]. The GPU based hardware side of the library implements block matching, belief propagation [13] as well as a constant space variant [78]. Of these latter three only the block matching is suitable for real-time implementation. OpenCV also implements several utility functions useful both for refining current implementations and for developing new algorithms. Examples include camera calibration, filtering techniques, feature detectors and other useful low level functions for data manipulation.

2.4.3 TBB - Intel Threading Building Blocks

Intel Threading Building Blocks (TBB) [62] is an easy to use C++ parallel computing library supporting multiple operating systems. It is built around the concept of creating tasks rather than threads, then using a multiple of parallel computing primitives large scalable parallel programs can be realized.

[1] http://opensource.org/licenses/BSD-3-Clause

The library dynamically assigns groups of tasks to available threads without a need for further programmer interaction allowing effective computation and ease of use. This dynamic nature also means that should a thread finish its work it will be assigned new tasks from threads with surplus tasks, thereby spreading the computational load and increasing overall effectiveness.

Chapter 3

Implementation

This section will explain some of the details of the evolutionary stereo algorithm parameter optimisation framework, which was built for this thesis.

3.1 An Overview of the Evolutionary Framework

An approach based on a genetic algorithm was picked as the chosen optimisation method as the individual behaviour and fitness landscape was unknown in advance making a more complex model difficult to construct. With genetic algorithm implementations typically only requiring a fitness calculation function and some control parameters, it makes for a good choice for this problem.

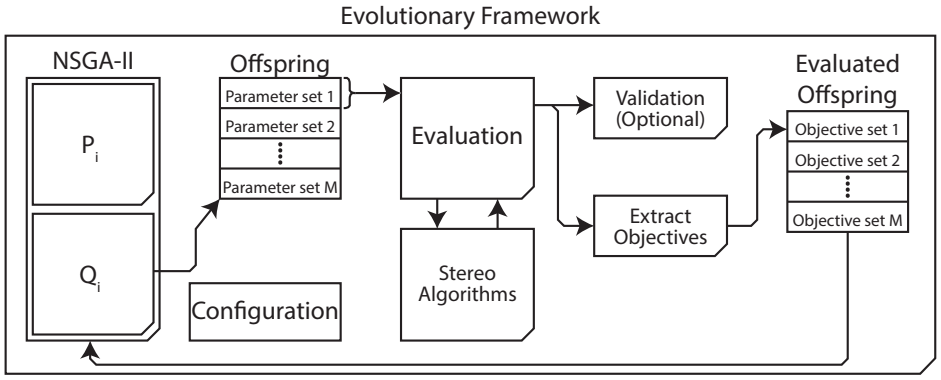


Figure 3.1: A general overview of the implementation.

As per Tippets et al. [72] it can be difficult to choose a trade-off between quality and speed as far as stereo algorithms are concerned. This would indicate that a multiobjective approach would work well as several metrics may be optimised simultaneously giving the user the ability to select their best solution after the fact. With this in mind NSGA-II was chosen as the EMO algorithm for this thesis, the selection of which is given in more detail

in Section 3.3.2.

Functions were built around this genetic algorithm creating an evolutionary framework tying it together with several stereo algorithms (Section 3.4) and the required support and evaluation functions necessary for robust testing. A simplified overview of this framework, as implemented in the C++ programming language, can be seen in Figure 3.1. After the initial framework startup and setup stage the program flow is dictated by NSGA-II with function hooks triggering the relevant framework functions at certain evolutionary steps including evaluation (Section 3.2), validation (Section 3.5) and run termination.

3.2 The Evaluation Method

For NSGA-II to know the performance of its generated offspring population it needs an evaluation function. This evaluation module rates each parameter set it receives on a selected data set containing image pairs of interest. A simplified overview of this function can be seen in Figure 3.2.

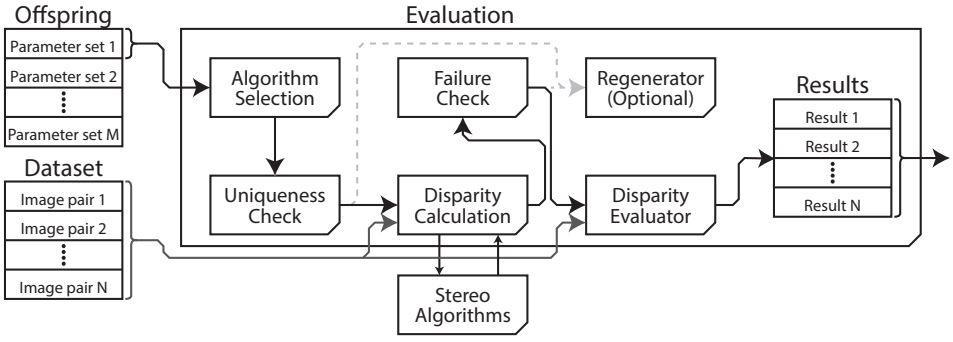


Figure 3.2: Overview of the evaluation approach used and the modules therein.

3.2.1 Choice of Data Set

The ability of an evolutionary approach to learn and generalise is in large part affected by the data set and its ability to sufficiently describe the problem. With a real-world focus potentially difficult situations for a stereo system becomes more common (see Section 2.1.6), this in turn provides an interesting challenge for the stereo parameters. Additionally the research of e.g. Vaudrey et. al. [75] has shown that good results on a synthetic benchmark does not necessarily transfer to real-world data. With this in mind the main contenders for the data set were the Middlebury 2014[1] data set and the KITTI[2] data set.

The Middlebury data set is interesting due to being a de facto testing standard in the stereo community. With the added robustness tests

[1] http://vision.middlebury.edu/stereo/data/scenes2014/

[2] http://www.cvlibs.net/datasets/kitti/

included in the 2014 version it challenges different aspects of stereo algorithms while providing very accurate ground truth data for evaluation. However, with only 23 image pairs available it may not be possible to both evolve good solutions and have data left over for verification, as such this set was only used during some preliminary experiments.

Unlike the Middlebury set, the KITTI data set is entirely focused on the real-world case. This comes at a cost of sparser ground truth images, but considering the increase to 194 image pairs this set should still contain the necessary data points required for a balanced evolutionary search.

The data set provides both colour and greyscale versions. Of these the greyscale version was ultimately chosen, both due to fair comparisons on the same basis as not all algorithms support colour information, but also due to the increased robustness of intensity information in comparison to colour data [27]. This choice may however reduce the capabilities of some of the additional pre-processing methods used in the implementation as object edges will be harder to read [32]. However, as it saves having to convert to greyscale during the evaluation, it saves a considerable amount of computational effort across all generated individuals. An example image pair with ground truth is presented in Figure 3.3.

Of the 194 total images only 40 images were reserved for training data. The main idea of reducing the training set size is to reduce the computation required in the evaluation step. However, while this choice will speed up each individual evaluation it is also likely to give increased variance in the results as the size does not adhere to the recommended 80% rule of thumb of the Pareto principle. Of these 40 training pairs three differently sized dependent training sets were created containing 20, 30 and 40 image pairs respectively. This was done to allow testing on the effect of the training set size on the resulting solutions. Two test sets of 65 and 89 images were created from the remaining 154 image pairs. Each image assignment was done via a random sequence generator[3] giving the image pair distribution for each data set as listed in Appendix B.

Data Set Implementation

The practical implementation of the data set is based around OpenCV image formats and their relevant library helper functions. Upon initializing the framework, the training and optional validation or test sets, are read to separate internal data set objects from their specified directories. The left, right and ground truth images are automatically stored and assigned as individual *tests* for later use in the disparity calculation and evaluation functions using this common test format.

3.2.2 Disparity Calculation

Before actual disparity calculations can start the parameters of the individual are sent to the selected stereo algorithm through a translation layer.

[3] http://www.random.org/sequences/

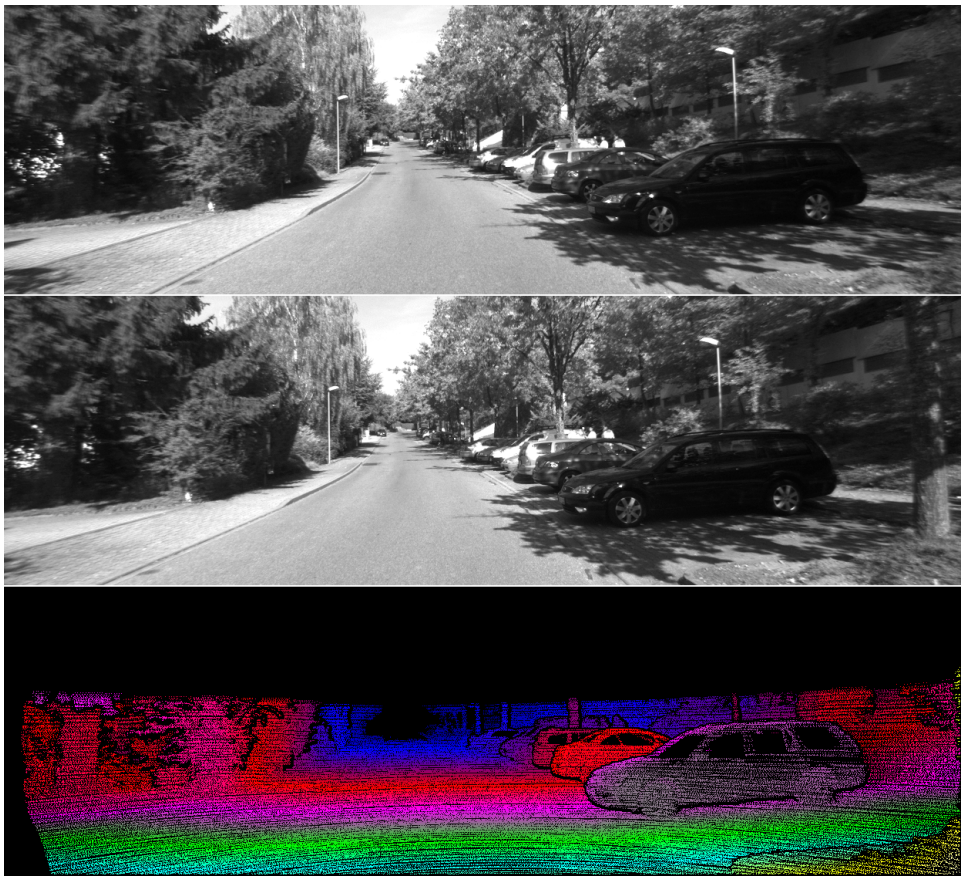


Figure 3.3: An example of the KITTI dataset left, right and ground truth images. Image pair 3_10 of the KITTI training set [20]. The GT has been colour mapped for extra visibility.

This layer maps the NSGA-II internal values into the algorithm with the option of altering the genome if illegal values are sent. The latter is used for parameters where certain values are incompatible with the algorithm, e.g. enforcing an odd valued window size parameter. While this makes for some unused parameter possibilities in the genome, the function was kept as it meant the NSGA-II internal state and output matched the parameter values actually used by the algorithms. This allowed a less error prone analysis of the parameters with the potential of easier verification of the found parameter sets outside the evolutionary framework.

After sending the parameters to the algorithm the remaining parameters, if present, are sent to the optional prefiltering module preparing it to alter the image pairs accordingly before the images are sent to the algorithm for processing. The actual prefiltering is handled just before the disparity computation starts allowing multiple operations at once if the parallel strategies of Section 3.2.4 happen to be enabled. The time used in this optional prefiltering step is recorded for later analysis.

Each algorithm is supplied with its own translation module providing a common interface to its disparity calculation function and ensuring that the

common image format of the framework is understood by the potentially incompatible format expected by the algorithm. This translation module is also responsible for reporting on any potential algorithm failures so that the disparity evaluator does not evaluate an erroneous image or even random memory causing non-deterministic quality metrics. An example of such an online failure would be if for instance the ELAS algorithm (see Section 2.2.4) didn't find enough support points to make a valid mesh with the given parameters, at which point it will terminate its calculation. Failing on one image skips the remaining image pairs and marks the individual as *failed* so that later modules can decide on what to do with these parameters.

For each disparity calculation the time taken is recorded and the total tallied up. If the total exceeds the current timeout threshold (See Section 3.2.5) further calculations are terminated and the individual is marked as failed. Should the disparity calculation succeed the output is wrapped in a common interface to allow for different disparity output formats and then handed over to the disparity evaluator in Section 3.2.3. Once all images are complete the tallied calculation time is stored with the combined evaluation results.

3.2.3 Disparity Evaluator

The disparity evaluator rates the resulting disparity images output by the stereo algorithms as compared to the corresponding disparity ground truth image contained in the data set. The actual disparity evaluation approach was originally based on the evaluation function used in the Middlebury v1.3 SDK[4]. Said function reports several quality metrics including the density, pixels outside a correctness threshold, and the sum of the latter two. These supported quality metrics were expanded on to include the RMS error and the worst data set error to give the following supported metrics:

- **Error** - The per pixel distance from the current disparity result to the ground truth. Used only internally in the evaluation method. Calculated for marked pixels only.
- **Bad** - A measure of quality giving the percentage of calculated pixels with Error greater than a certain disparity threshold.
- **Invalid** - A measure of result density returning the percentage of unmarked pixels in the disparity result. Each algorithm has a special *uncalculated* disparity value which is stored for uncalculated or removed results and reported to the evaluation function once that pixel is requested.
- **RMS** - A quality metric giving the Root Mean Square of the above Error across the entire result. Calculated for marked pixels only.
- **AvgErr** - The average pixel distance between the calculated disparity image for marked pixels and the ground truth.

[4] http://vision.middlebury.edu/stereo/submit3/

- **MaxOut** - The worst $Bad + Invalid$ result for this parameter set across all images evaluated.

Equations for the Bad, Invalid, RMS and AvgErr metrics can be found in Section 2.2.1.

To ensure compatibility with the remaining framework the evaluation method was adapted to use a common disparity retrieving interface allowing different stereo algorithm output formats to be evaluated using the same method. The KITTI data set contains two different ground truth images, one for all pixels and one containing only the non-occluded areas. Of these the non-occluded was used for the evaluation module which required a small change to support the different KITTI format.

Having evaluated the disparity output a Result-structure of all the reported quality metrics is created for each image pair and combined with the runtime data from the disparity calculation. After which, it is tallied up within the framework for use in later file output, objective creation and potential validation steps.

3.2.4 Evaluation Computation Strategies

During early parameter testing the initial serial approach to evaluation proved to be too slow, with generations taking hours and complete evolutionary runs several days. While this serial approach was simple to implement and provided a good performance baseline, the slow evaluations made testing tedious and time-consuming. With this in mind two parallel computing approaches, as illustrated in Figure 3.4, were developed to help speed up the process.

	Max Threads	Synchronization Required?
Serial	1	None
Parallel A	Population Size	None
Parallel B	Dataset size	When combining results

Table 3.1: A comparison of the serial and the parallel computational methods used in this project.

Table 3.1 shows a comparison between the implemented methods. Parallel method A initially splits the population amongst threads then each evaluates the complete dataset on its assigned individuals. As individuals are very different in computational complexity some threads will finish their work earlier than others, at which point they will opportunistically take available tasks from the other threads as per the Threading Building Blocks (see Section 2.4.3) implementation. This is expected to provide very good utilization of the computing resources available. It is also expected that the population size will be larger than the dataset size allowing a greater number of maximum threads.

Parallel method B takes a different approach in that one individual is selected, then the dataset is spread amongst threads and evaluated for that individual only. Once complete the next individual is fetched and the

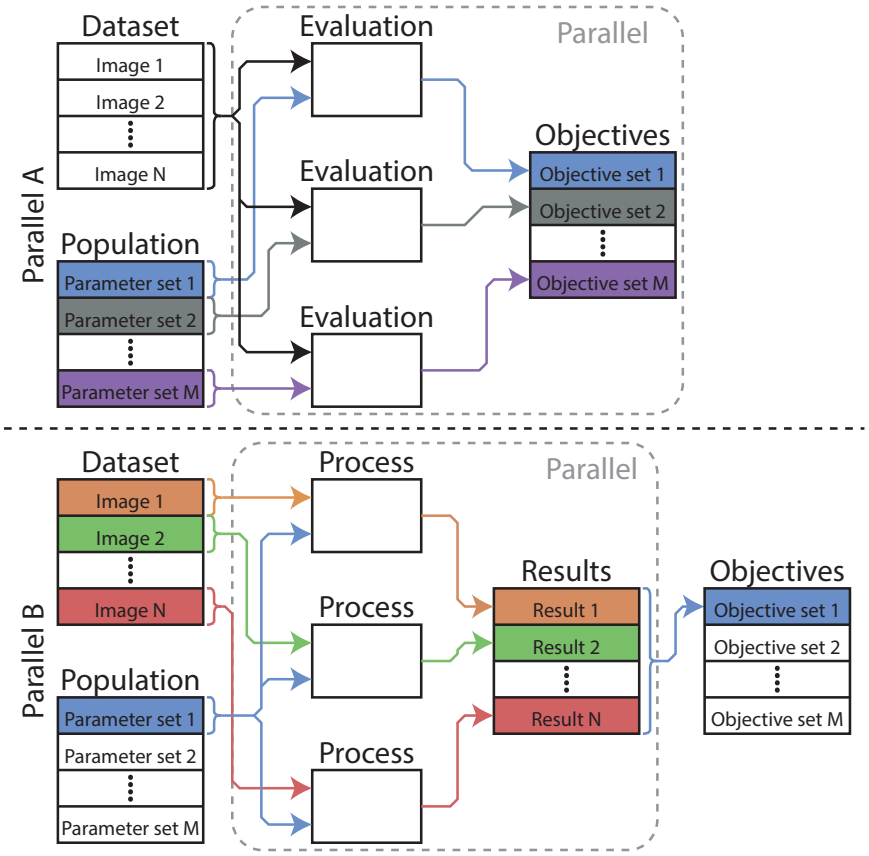


Figure 3.4: A simplified overview of the parallel evaluation strategies used for increasing evaluation speed. The Evaluation node of Parallel A includes the steps shown in Evaluation as well as the actual objective extraction. The Process node of the Parallel B strategy includes the Disparity Calculation, Failure Check and Disparity Evaluator of Figure 3.2.

evaluation continues. This limits the number of threads to the dataset size and there is more overhead in thread communication, starting and stopping than the A-method. Additionally, for each individual it is likely that some threads will go idle at the end of each evaluation providing less utilization of the processor. However as all individuals are subject to this effect it is expected that if runtimes are compared that this method will provide more stable results rather than biasing just the last couple of individuals.

A third method (C) was also considered wherein each thread was given the entire population and a subset of the dataset at the start of the population evaluation. This could be thought of as an extension of the B-method with greater data locality and less thread creation and start/stop overhead.

These methods will later be compared in regard to fairness and speedup in comparison with the serial evaluation method. Said speedup will be calculated in accordance with Equation (3.1).

$$Speedup = \frac{Time_{old}}{Time_{new}} \tag{3.1}$$

3.2.5 Timeout methods

Certain parameter values in combination with others may produce especially slow individuals causing the time for each evolutionary run to spiral upwards as similarly slow individuals may be produced as offspring. The problem is illustrated in the recorded runtimes of Figure 3.5, which if multiplied by the dataset size and the number of total individuals would lead to very slow progress indeed. As part of the slow parameters may be useful on their own, restricting the parameter range was not seen as a good option in controlling this potential problem. A good early solution was to include the runtime as an objective, but when testing with different objective types this was not always an option. Two more direct solutions were implemented to control this issue via an internal evaluation timeout mechanic.

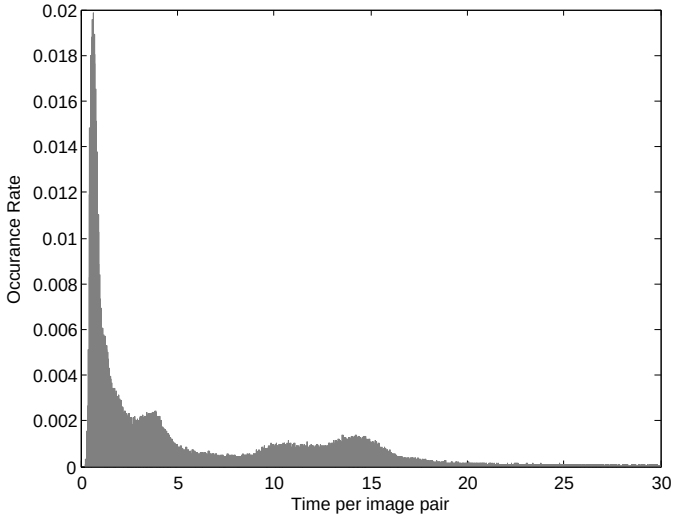


Figure 3.5: A typical runtime distribution seen for the ELAS algorithm if neither a runtime objective nor a timeout method is used.

The Hard Timeout Method

The first method implemented a hard timeout wherein individuals using more evaluation time than a certain threshold were terminated causing the remaining images in the dataset to be skipped. As the missing evaluation steps would generate illegal results, a decision would then have to be made in regard to what was to be done with the slow individual. Two strategies were implemented:

- *Reject* - Assign worst-case objectives preventing the individual from propagating its genes to future generations.

- *Regenerate* - Ask NSGA-II to create a new random genome within the allowed limits, then evaluate these new parameters. Should this too fail the Reject strategy is used to prevent unfortunate loops.

The timeout threshold was typically set some limit above the desired maximum runtime based on the assumption that individuals exceeding the actual desired runtime may still contain pieces of genetic information useful to the search.

The Soft Timeout Method

The second implementation is based on a soft threshold revolving around the constraint violation mechanic of the NSGA-II implementation used, wherein the non-dominated sorting is based not only on dominance, but on how well the individual's parameters were compatible with the evaluation step. This allows the evaluation to add a penalty to each individual exceeding a certain runtime threshold scaled to the exceeding amount. An expected benefit of this method is better convergence in runs with a high degree of timeouts, as in this case the individuals are kept rather than in effect removed. It may also be possible to set the soft threshold to a value closer to the desired maximum runtime without fear of losing good genes barely exceeding the threshold. This method does not preemptively terminate the evaluation of individuals, but was instead used in conjunction with the hard timeout set to a significantly larger value to catch any unreasonably slow evaluations stalling the run.

Timekeeping

The timing method used was initially based on the C++ clock()-function based on the belief that it was based on the CPU thread-time and hence somewhat robust to the actual system load. This implementation was inherited from an early Linux version of the project which was later abandoned due to compilation issues caused by user access restrictions. As it turns out the compiler implementation of the clock-function is entirely different in the Windows-based Visual Studio 2013 C++ compiler, with it implementing the function as based on the real-time used rather than the CPU time used. For a computer under heavy load this means that the recorded time in large part becomes unreliable as the time between task start and end does not properly indicate the time a task was actually actively calculating, due to potential task switching. In turn, this caused the timeout method to occasionally terminate a simply unlucky individual rather than an actual slow one. This issue affected some of the early experiments, but was later corrected once these experiments highlighted the issue.

3.2.6 Uniqueness Constraint

The exploitation step in a heuristic search creates future parameter sets close to known good solutions in an effort to refine future solutions. This increases both the likelihood of hitting the same combination of parameters

and of getting similar resulting objectives more than once over the duration of the search. Algorithms controlled by a small number of only discrete parameters are more exposed to this problem due to the size of the search space. As such both the BM and SGBM implementations (see Section 3.4) are heavily affected with a high frequency of duplicates.

Individuals similar in objective space are discouraged from propagating by the NSGA-II crowding distance metric, but for this to happen an evaluation must first take place. In the case of parameters duplicating previously generated solutions this step represents a significant wasted effort due to slow evaluation. Additionally the crowding distance metric is only used in cases the entire front won't fit, which causes duplicates to survive until the front is either changed, fully superseded, or split during survivor selection. As such it may be desirable to filter parameter duplicates in advance.

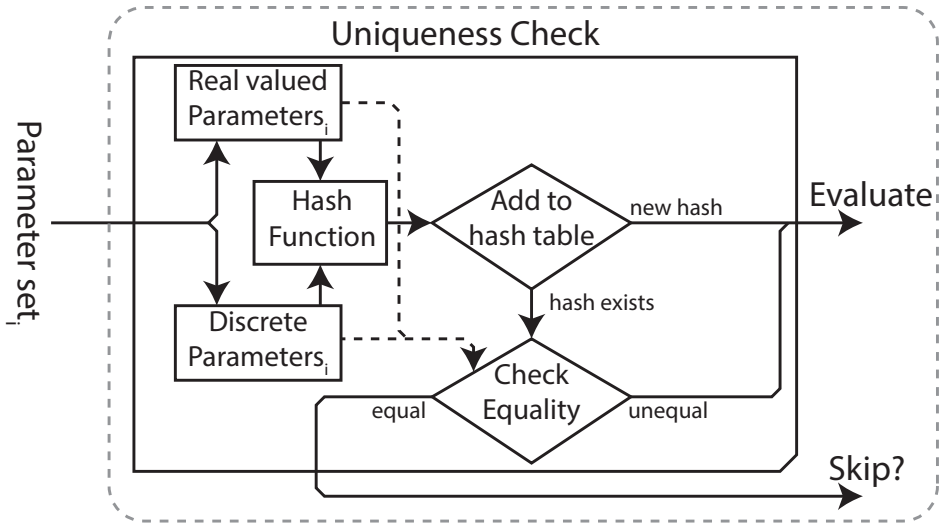


Figure 3.6: An overview of the uniqueness checker as implemented in the population evaluation function.

To account for this problem a uniqueness check was implemented in the population evaluation step as per figure 3.6. Once each offspring is sent to evaluation their parameters are compared to all previously evaluated individuals and skipped should the parameters match any previous individual. Such a comparison against all evaluated individuals would be computationally expensive, hence a hash lookup is performed based on a hash-value calculated from both the position and the value of each parameter. Should the value not be present in the table the individual is immediately allowed to be evaluated as it has not been previously seen. If such a value is already present either the value has been seen before or more than one parameter set has been mapped to the same value in a hash-collision. An equality check is performed to determine which is the case with discrete parameters directly checked and real parameters assumed equal if within a λ of each other. Should all parameters be considered equal

a decision has to be made amongst the following options:

- *Keep* - Allow the individual to be evaluated based on the assumption that repeats may be good genes, giving this genotype additional priority in further selection.
- *Reject* - Pro-actively skip the evaluation to save time and assign a worst-case score to greatly reduce the likelihood of that individual making it to the next parent population.
- *Regenerate* - Block the individual from evaluation, randomly regenerate its parameters from scratch, then evaluate it. Thereby providing new genes and increased diversity to the population.

The latter approach is similar to the Clone Management Principle of Mandal et al[48], but in this case it is simplified to making the decision in the parameter space only.

3.3 Evolutionary Setup

This section describes the objective functions used, the setup for the NSGA-II portion of the framework and how the experiments were conducted.

3.3.1 Objective functions

For the NSGA-II approach to work objective functions will need to be created to rate the fitness of an individual thereby giving the genetic algorithm a way to compare solutions and work towards the pareto front.

The framework supports several objective functions configurable in type and number via the command line. For each image in the dataset each evaluated parameter set will generate a results object which is averaged in the framework giving the following potential selectable objectives:

- **Out** - The average number of total outliers in the results. An outlier is considered either a Bad or Invalid pixel, hence $Out = Bad + Invalid$.
- **Runtime** - The total runtime of an individual parameter set summed up of the Disparity Calculation time and the optional Prefilter time. (Section 3.2.2)
- **Bad, Invalid, RMS, AvgErr** - The mean values of the Disparity Evaluator quality metric outputs. (Section 3.2.3). The Bad objective is equivalent to the Bad in the Middlebury benchmark and the Out-Noc in the KITTI benchmark.
- **MaxOut** - The worst $Bad + Invalid$ calculated for this parameter set across all image pairs.

The Out objective was much used in early experiments as this would handle quality and density both. However this would say little in regards to the individual contribution of each effect and was later split into

multiple objectives for better exploration [36, 42]. This change also set the population examination standard as most pareto front plots will be in the Bad-Invalid objective space.

The Bad objective required a threshold to be set on how far a result could be from the ground truth before a disparity was deemed to be wrongly calculated. Based on the current KITTI ranking this threshold was set to 3 disparity levels. Converting this disparity to its equivalent distance gives the accepted error ranges in meters as in Figure 3.7. Any disparity calculation with a result below the line is accepted by the 3 disparity threshold, while any above will be marked as bad and tallied in the Bad objective.

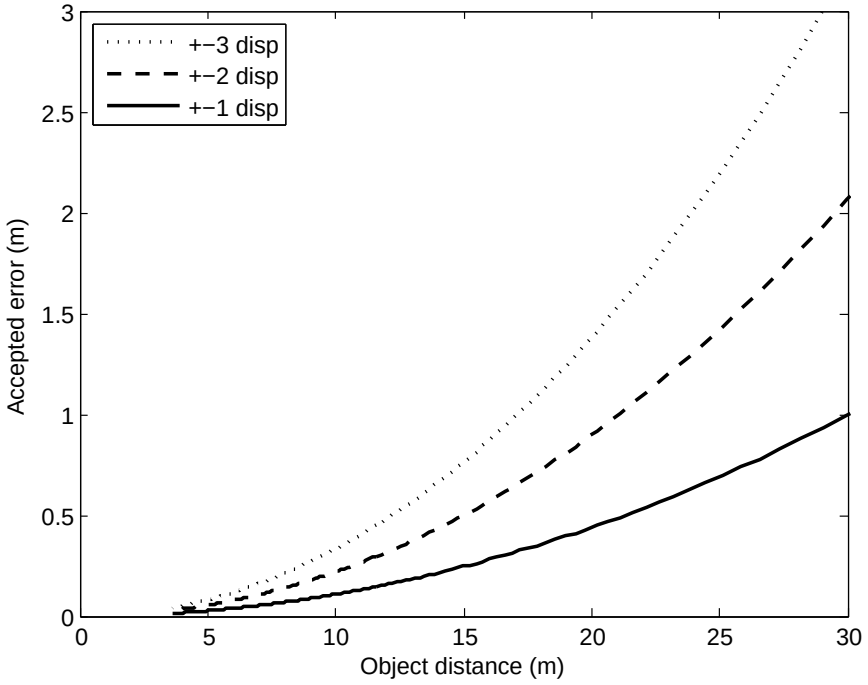


Figure 3.7: The accepted error given a certain disparity threshold of the Bad objective.

Objective selection is limited to 1-3 objective targets, as more than 3 objectives is disallowed due to the large search space causing slow NSGA-II progress due to rapidly filling rank 1 with the entire population [10].

3.3.2 NSGA-II

For the evolutionary multi-objective approach of this thesis a suitable EMO algorithm had to be picked. Of the sample of possible choices outlined in Section 2.3.2, NSGA-II was picked based on its popularity, known good performance on unknown problems and the small number of required control parameters. NSGA-II is further detailed in Section 2.3.3.

Given the objectives mentioned in Section 3.3.1 it should be possible to include both a density and quality objective to match the most commonly listed quality metrics on official benchmarks, while leaving room for

an optional runtime or helper objective [36] within the practical three objective limit of NSGA-II. Any more and a NSGA-III [10] implementation may have been necessary for good results, but the official source code for this variant was not available at the time of this thesis.

The popularity of NSGA-II means that finding a suitable implementation is easier than less used or more esoteric variants. As the chosen stereo algorithms (Section 3.4) all had C and C++ implementations, finding a similarly programmed NSGA-II implementation meant there was no need to re-program the wheel, and more time left over for the evolutionary framework and its experiments. The version used was a C++ conversion, as implemented by David Ojeda [58], of the original public KanGAL NSGA-II implementation.

This implementation featured all the latest NSGA-II characteristics including objective scaling, constraint handling, real and binary genes, accessible function handles for custom evaluation functions, and file handling including saving and loading of current progress. In addition multiple example programs made implementation into the framework easier.

3.3.3 Setting the crowding distance type

The crowding distance measure of NSGA-II is used to ensure diversity within the population of individuals as previously described in Section 2.3.3. The selected implementation of NSGA-II supplies two normalized crowding distance operators giving the option of basing itself on either parameters or the objectives themselves. Said normalization step ensures that when multiple objectives are used, the range of each objective does not change the weight of that objective in the crowding distance calculation.

Crowding on parameters

To better explore the parameter-space one method is to enforce dissimilar parameter choices by setting the crowding distance to be based on the algorithm parameters alone. This prioritizes parameter diversity over phenotype diversity with the goal of more evenly testing parameter combinations.

However, some parameters are dependent on each other and may not contribute to the solution in certain cases due to being effectively disabled further up the parameter chain. In this case multiple phenotypically identical solutions are kept causing the search to slow down. An example of such problematic parameters would be various additional filter constants should the filter itself be disabled. An additional, but greater problem is how the crowding distance of each extreme of each parameter is set to a guard-value to guarantee selection. This in essence means that up to $2 * parameter - count$ individuals in each rank will contain the infinite crowding distance guard. For most reasonable population sizes this entails that most of the population will have infinite crowding distance, and as these values are given selection preference (see 2.3.3), the metric will

actually remove non-extreme parameters during selection should the front not fit in its entirety. In essence, enabling this feature leads to a similar dimensionality issue as increasing the number of objectives normally does, causing problems for NSGA-II once more than three are present.

Additionally the crowd on parameters function as implemented in this version of NSGA-II only applies to real valued parameters and not both real valued and binary represented parameters. This causes a compatibility issue with the current evolutionary framework and makes the method unsuited for this application.

Crowding on objectives

The more traditional crowding distance measure based on inter-individual distance in the objective space is also supported by this NSGA-II implementation. As it does not suffer from the same drawbacks as the parameter space implementation, it was chosen as the default crowding distance measure in this thesis.

3.3.4 Selection, Mutation and crossover

The chosen NSGA-II implementation uses the simulated binary crossover (SBX) operator for real valued genes and a two-point crossover operator for binary genes. The specific of these operators are described in more detail in Section 2.3.1. The crossover parameters were left at the default implementation recommended values, as the evolutionary stereo parameter problem was not known well enough in advance to make further judgement. This meant a crossover probability of 1 for both methods and the SBX distribution index η_c left at 10.

For mutation, this implementation uses the polynomial mutation operator for its real values and binary bit mutation for the discreet ones. These operators are outlined in Section 2.3.1. The mutation probability was locked at the default 3% and unchanged for the remainder of the experiments even if a different value may have been better for the larger population sizes and the specific genome sizes or each test.

3.3.5 Added and changed features

While the NSGA-II implementation was largely used as is, some of the framework features required small changes for ease of implementation.

To better test certain genomes within the framework, the possiblity of setting the initial population was added. While not used for the evolutionary runs, this feature allowed easy retesting of parameters suspected of stereo algorithm crashes during setting of the initial parameter ranges. In addition it allowed testing of the genome representation by inputting known good parameters and checking whether the generated phenotype matched the origin. The latter uncovered a problem with the binary represented part of the genome which led to an incorrect mapping and required a small change to the NSGA2 implementation.

Beyond these changes, two technical alterations were done to reduce reliance on further external code libraries and reusing the ones used in OpenCV (Section 2.4.2) and its stereo algorithms. This included a change from OpenMP to TBB (Section 2.4.3), matching the threading library used in OpenCV. The random number generator was also changed from the Boost library to the equivalent C++11 Mersenne twister algorithm.

Soft non-dominated sorting for non-deterministic objectives

The original non-dominated sorting (NDS, see 2.3.3) sorts evolutionary objectives based on a direct comparison in which the smaller is better. This works well for deterministic objectives, but non-deterministic objectives like runtime are unfairly treated by this hard threshold, as even the same parameters evaluated twice will not get the same objective output. To better account for this unfairness a novel soft threshold NDS was developed, in which a non-deterministic objective has to be better by a certain percentage before it is truly considered superior. Figure 3.8 shows this concept. The scale factors are set on a per objective basis allowing deterministic objectives to keep a hard threshold while non-deterministic ones may individually select appropriate values. This was implemented as an optional feature to be enabled in certain tests.

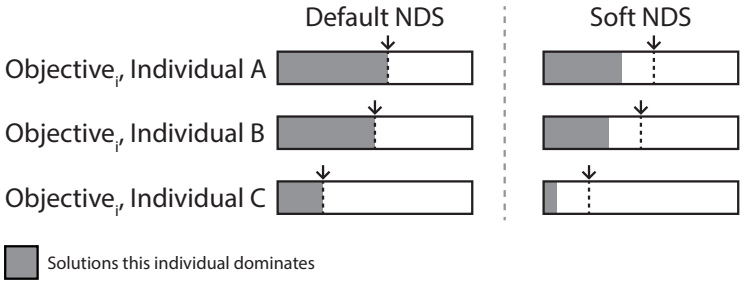


Figure 3.8: Comparison of the default non-dominated sorting with the soft non-dominated sorting presented for non-deterministic objectives. In the default NDS individual A has a better objective than both B and C, whereas individual B dominates C. For soft NDS individual A no longer dominates B as it is close enough to be considered an equal.

3.3.6 Run Setup and Worker Assigments

A parameter optimisation run starts by specifying a desired configuration through a command line interface which controls the stereo algorithm used, evolutionary parameters and the behaviour of certain extra modules within the framework. For each experiment a batch script method was used to automatically do multiple runs and save the framework and NSGA-II output to directories relating to the test at hand. This allowed easy analysis after the fact using Matlab as a statistical tool.

With multiple runs used per experiment and the cost of evaluation (see Figure 3.5), many runs on several experiment configuration would take

too long on a single worker machine. In this regard two methods were used for combining multiple workers in making several evolutionary runs at the same time. The main two methods are illustrated in Figure 3.9. The first method assigns the same experiment variation with all its runs to the same worker. The idea of the variation reduced method is to spread the experiment type across multiple workers thereby reducing the influence of worker performance on the results. While all workers were powerful machines and supported the same SSE4.2[5] instruction set, some had slightly different architectures causing variation in the resulting output when a timeout method or runtime objective was used.

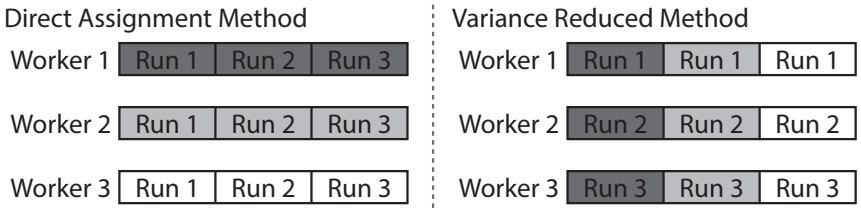


Figure 3.9: Two computational strategies for multiple runs on three different experiment variations, indicated by the different greyscale boxes. Three runs are done on each experiment in this illustration.

3.4 The Stereo Algorithms

Several stereo algorithms were considered for addition to the evolutionary framework. Criteria for inclusion within the framework included: available source code, short runtimes, good results on the KITTI benchmark, centralized behaviour control through parameters, and ease of conversion with few external code library requirements. Additionally it was desired to have a variety in the selected stereo algorithms, both within their inner workings as well as their number of control parameters, allowing a more thorough testing of the evolutionary framework.

With this in mind, the KITTI benchmark pages[6] were sorted by results and available source code, then a runtime threshold of 2s was applied to filter out candidates with too costly evaluations. This gave the following candidate algorithms in order of current score from best to worst: SPS-st, rSGM, OpenCV-SGBM, ELAS, OpenCV-BM-post and OpenCV-BM.

Of these the rSGM would have been interesting due both to being based on the popular SGM algorithm, but also due to it using the census transform for matching costs which is known to provide robust results[80]. However, the 0.2s time in the KITTI results table is the time used in parallel striped mode wherein multiple CPU or GPU threads are used simultaneously to calculate parts of the resulting image. With the evolutionary framework already calculating multiple instances in parallel no particular real time

[5] https://software.intel.com/sites/default/files/c8/ab/17971-intel_20sse4_20programming_20reference.pdf

[6] www.cvlibs.net/datasets/kitti/eval_stereo_flow.php

speed-up is to be expected with this addition as all computational units are likely to be busy. As such it is expected that this stereo algorithm would behave similarly to other SGM methods and be on the slow side when used in this evolutionary context, thereby limiting the number of possible runs.

The ELAS algorithm (see Section 2.2.4) was included as it was featured in the Middlebury SDK and was thereby used for much of the initial testing and framework development. Controlled by a large number of parameters, the ELAS algorithm gave a very wide performance variation which made it a good choice for optimisation. It also quickly highlighted the problem with expensive evaluations as far as runtimes were concerned making for a stricter stereo algorithm selection and ruling out the SPS-st algorithm based on its posted time.

Of the remaining OpenCV (Section 2.4.2) based algorithms the regular BM (see Section 2.2.2) was chosen over its OCV-BM-Post variant, as the latter turned out to have a broken source code page. While not the best contender on quality it is a very fast algorithm and represents a more traditional window based stereo approach. Having included the OpenCV library its SGBM algorithm (see Section 2.2.3), a block-matched SGM variant, was also implemented in the framework as a more robust dynamic programming approach.

3.4.1 Genome

Upon initialization of the framework the interface of the active algorithm is queried for its control parameters, their count and size as well as their allowed range. These are then encoded into the genome representing the parameters.

As the NSGA-II implementation supported both real valued and discrete genes, the stereo algorithm control parameters were divided based on the internal numeric type used. Representing the integer parameters as binary helps in reducing the search space, ensures a more direct and simpler to follow genotype→phenotype mapping, and allows more effective pruning via the Uniqueness checker mechanic (Section 3.2.6).

3.4.2 Setting the Parameter Limits

When decoding the genome, constraints have to be added to make sure the generated parameters are within the legal range of the values accepted by the active stereo algorithm. The parameter limits for this procedure were initially found based on the limited documentation available, and the source code of the stereo algorithms. Even so, early ranges were too broad and led to occasional crashes due to illegal parameter combinations. To account for algorithm crashes the parameters of the offspring population were written to file before evaluations started so that the problem could be traced and accounted for in updated limits. This was able to detect a rare problem with the ELAS internally generated planes, which given the right parameters and image content could cause an illegal memory access and crash the algorithm. This required a small algorithmic change and

showed the unintended potential for using the evolutionary approach for semi-randomized algorithm testing.

After initial testing with very wide parameter limits, the ranges were reduced to cover a large range around the final parameter values used in these initial pareto fronts. The goal of this reduction was to reduce the search space and thereby help future evolutionary efforts. These legal ranges are described in Appendix A as the *Initial set*.

3.4.3 Additional Prefiltering

To get a better idea of the performance of the solutions they would have to be compared with published results. The results on the KITTI benchmark typically feature a prefiltering step to better account for noise in the image pairs and thereby improving the result. An example of this would be the gaussian smoothing added as a preprocessing step to the published ELAS algorithm results. Testing the ELAS Middlebury default parameters on Test Set B with and without its Gaussian prefilter showed a 0.77 change in the Out percentage with a negligible change in runtime once prefiltering was added. Hence for fair comparison several prefiltering types were added to the project. This also serves the dual purpose of testing how an algorithm change can affect the output and how the framework handles this effect.

There are several kinds of noise and problems present in real-world data which may benefit from pre-processing techniques to make the image pairs easier to understand for the stereo computations to follow. As examples images may have additive and multiplicative noise or the pairs may even be unevenly illuminated making matching more difficult.

Prefilter	Brief description
Bilateral	Edge-preserving smoothing [1].
BilSub	Background subtraction using the Bilateral filter [28].
Gaussian	Gaussian smoothing
MeanGaussian	Mean + Gaussian filtering
MeanLoG	Mean + Laplacian on Gaussian smoothing and edge enhancing [69].
Scharr	Edge extracting, illumination invariant
MeanScharr	Mean + Scharr
Domain	Edge-preserving smoothing
Guided	Edge-preserving smoothing [32].

Table 3.2: List of prefiltering types available to the framework.

Table 3.2 gives a brief overview of the filters included in the framework. The Bilateral, Gaussian and Scharr filters are unchanged OpenCV filters. Domain and Guided filters required an optional OpenCV module, but are otherwise unchanged. The Mean* filters are the internal framework names for custom filter combinations based on the guideline set by Shuchun [69], wherein several filter types are combined to handle different aspects of the noise. The histogram equalization in the paper was not applied as it was difficult to implement without negatively affecting the matching step of the algorithms.

Edge enhancing prefilters like the step used in the LoG and Scharr filters are usually applied to make the preceding matching independent of illumination differences in the image pair. This means that the output of the filters should be used directly as the input to the stereo algorithms. However as some of the algorithms tested have their own prefiltering methods, doing so could create a conflict and reduce the amount of useful information remaining in the images. As such a mixing ratio scheme was implemented in which the amount of edges and source image could be controlled (Equation (3.2)). An added feature of this α mixing ratio is that it allows NSGA-II to effectively turn off this part of the filter if it is deemed unnecessary by evolution. A similar technique was also extended to the mean filters, where a sufficiently large mean threshold disables the filter. In turn, this will make it more obvious whether a newly found great result was the product of the tested filtering technique or simply a fluke find of a new global best minimum.

$$Image_{filtered} = \alpha * Original + (1 - \alpha) * Edges \tag{3.2}$$

Evolving the actual filter type was considered as an option to better visualize the potential pareto front the combined filtering techniques and algorithms are capable of. However, with the filters each using from one to six control parameters this approach would require six evolveable parameters to simultaneously account for each and every filter as the number of parameters must remain constant across the population for the chosen optimiser. For many of the filters this would mean that several of the parameters would end up carrying no meaningful information. As the number of total parameters define the search space these extra parameters are likely to simply slow down the search. Additionally, as the uniqueness checker assumes all parameters alter the result, two individuals with essentially the same parameters with the exception of non-used filter parameters would still be counted as different and wastefully re-evaluated. Instead the program was modified to allow filter type to be specified in advance with the option of evolving both algorithm and filter in unison, or running evolution on the prefilter only using known good algorithm individuals. This approach requires more runs in total and more work if a combined pareto front is to be created in the end, but an added benefit is that the quality and variance of the filtering techniques themselves may be more easily compared. Once a run is started the selected prefilter is queried for its parameter types and their legal ranges, which are then appended to the genome limits sent to the NSGA-II initialization. This is illustrated in Figure 3.10

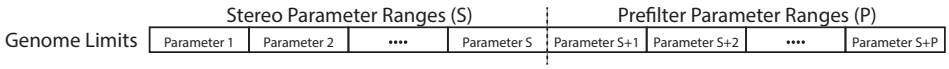


Figure 3.10: An illustration of how the prefiltering parameter limits are appended to each individual genome. This process is the same for the real-valued and binary represented parts of the genome.

Whether the prefiltering step is worth it or not depends both on the quality of the new result and on the extra overhead attached to the technique. As such the amount of time, both in total and the part used by the prefiltering, is kept track of so that such a judgement can be made on the technique's usefulness.

Configuring OpenCV

Several of the above filters or the parts thereof, are available in the OpenCV library (see Section 2.4.2). For added support of guided and domain transform filters, the optional Extended Image Processing (ximgproc) module had to be compiled in. To add this extra functionality a custom OpenCV version was created using Cmake[7] and Visual Studio as outlined in the procedure on the OpenCV optional module source code depository[8]. This was based on the state of the OpenCV source code depository as of 2015.02.02.

To better test different prefiltering strategies the OpenCV Block Matching algorithm was altered slightly to allow its internal prefilter to be disabled should the evolutionary approach find it to conflict with the prefiltering techniques in this section. This small change was however not possible to replicate in the SGBM algorithm as its prefiltering is built into the entirely uncommented cost function. With additional internal preprocessing applied to the external prefilters it is possible that SGBM will benefit less from these techniques.

3.5 Population Validation

As the search progresses it is desirable to know not only the increased success on the training set, but also verify the results on an independent dataset. This tests the ability of each solution to generalize and not overfit to the specifics present in the training data.

The current system allows selection of the validation set used. By default the independent 65 image set as described in Appendix B.2.2 was used for this purpose. Note that the increase in size over the training set would cause undue computational effort should the entire population be re-evaluated anew. As such the number of validated individuals must be reduced to a smaller portion of the population size.

With this in mind three main methods were implemented. The two first methods are online, meaning they work during the search effort. Of these, one evaluates only the best N individuals on a set objective, the other re-evaluates the entire rank 1 of the population. The output from both methods contains the evaluation results, including all the possible objectives regardless of the currently active ones. This allows later comparison of the individuals outside of the bounds set by the active

[7] http://www.cmake.org/

[8] https://github.com/ltseez/opencv_contrib/

objectives. Lastly the third method validates the population in an offline manner, by looking at the whole population after the run has completed.

3.5.1 The N Best Out Percentage Method

This external validation method keeps a running tally of the best N individuals as ranked by their Out metric (see Equation (2.3c)), regardless of the currently selected objectives. By keeping more than the very best individual the method allows a choice at the end of each run, ie. if a small loss in Out could allow a more sizeable gain in runtime. The choice of a static objective is likely to be a useful attribute when comparing search progress for different objective strategies as they would otherwise be more difficult to compare, as the NSGA-II output by default only includes the current objectives.

As each individual completes their evaluation on the training set, their full results and the parameters which garnered them, are offered to the validation module and accepted should the Out objective be better than the individuals currently held. Once the entire generation has finished evaluating, any *new* individuals in the validation module are re-evaluated on the validation set. Additionally, each generation, the various quality metrics on these individuals, as gathered from both the training and the validation set, are output to file for further analysis.

3.5.2 The Online Pareto Front Method

While the Best N method is useful for comparing different objective functions, it is not so useful when multiple runs with the same objectives are to be compared, or if the validation movement of the front is to be looked at each generation. In these cases a full validation of the entire pareto front would be useful.

Tying into the population report function of the NSGA-II implementation allows this method to do exactly that. The report function is presented with the parent population after survivor selection and allows direct access to the internal individual representations needed to extract the rank 1 individuals relevant to the method. Once found the individuals are compared to the previous rank 1 individuals stored in the validation method and any new entries are re-evaluated on the selected validation set. To start with this presents a large increase in evaluations as large front movements are to be expected, but as the search converges and only small front changes are made, little overhead is incurred. The rank 1 results from each generation is output to file with all quality metrics present per individual allowing further analysis.

3.5.3 The Offline Pareto Front Method

Once the first rank of NSGA-II occupies more than the entire population size, NSGA-II is required to simplify the front based on crowding distance so that it will fit, as such the final front does not fully describe the search

space covered. This effect is illustrated in Figure 3.12. To get the entire front an offline pareto front tool was implemented in Matlab searching through file output of the evaluation routine to find the total front after the fact. Writing the parameters of the selected pareto optimal individuals to file allows the framework to re-evaluate them on a desired verification set which is useful for both test and validation. As an offline method, fronts not directly correlated with the set objectives can also be constructed. An additional feature of this method is the ability to aggregate multiple runs into one total front thereby combining their strengths into what should be an even more well defined front. The workflow of this method is illustrated in Figure 3.11. Combining several runs happens within the Matlab portion by inputting multiple evaluation files which can each have their fronts verified independently or combined into a total front. Once verification results from the framework are complete the final output can be read back in to Matlab for further comparison and analysis.

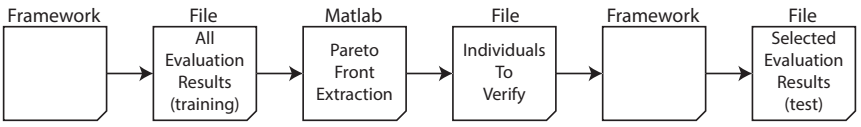


Figure 3.11: The workflow of the Offline pareto front method.

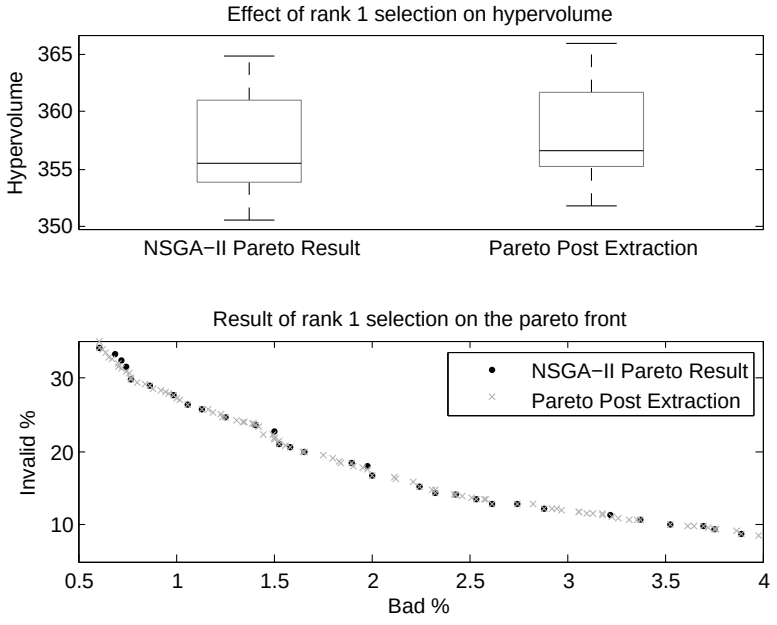


Figure 3.12: Extracting the pareto front post run using the offline method. Top plot shows a boxplot of 12 runs with the hypervolume calculated based on the NSGA-II output on the last generation and the equivalent based on the entire population. The bottom plot shows an extract of part of the pareto fronts showing the difference in detail and how NSGA-II spreads its selected solutions. Data from the Timeout experiment in Section 4.5

Pareto Front Search Space Reduction

When a combined pareto front across multiple runs is sought, a great amount of calculation is required to extract said front. Initially only a slow, presumably $\mathcal{O}(N^2)$ algorithm was used, as the faster Matlab functions available through the Mathworks file exchange website[9] required the C to Matlab MEX[10] function, which had no available compatible compilers on any of the computers used in writing this thesis. To get results in a reasonable time a method was required to split the search space into smaller chunks so that the slower pareto front algorithm was able to calculate the results in reasonable time.

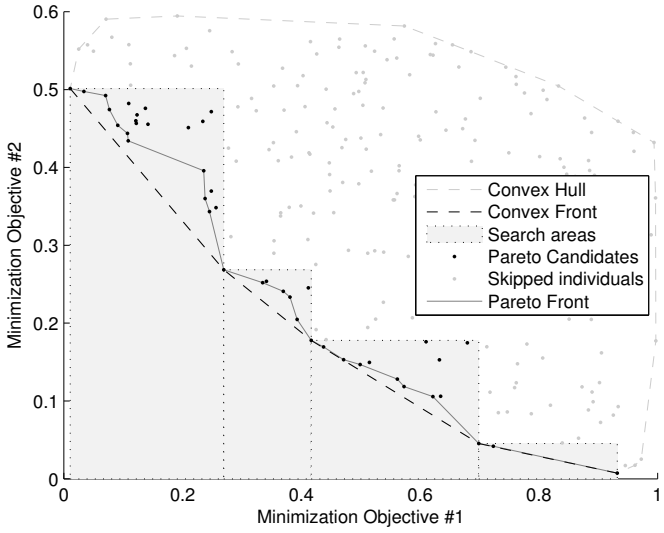


Figure 3.13: An overview of the technique used by the pareto front search space reduction method as implemented in Matlab.

Dividing the search space was done based on the convex hull algorithm. For the two-objective case this algorithm can be simply described by imagining drawing pins placed on a board in relation to each individual objective pair, then wrapping a rubber band around the objective space. Upon release, any pin touching the rubber band exemplifies the outer perimeter of the population and hence describes the convex hull. If we move along this perimeter the two objective values will each change direction twice allowing the convex hull to be divided into four distinct quadrants if split at these points. As the current NSGA-II version assumes minimization objectives, in this case the interesting convex hull quadrant lies closest to the origin. In a related property to the pareto front all points on this part of the convex hull will also describe the convex part of the pareto front itself.

This latter feature has seen some use as an alternative metric within NSGA-II [44, 55], due to it being easier to compute than the typical pareto

[9] www.mathworks.com/matlabcentral/fileexchange/
[10] www.mathworks.com/help/matlab/ref/mex.html

front. However in this method the previously mentioned related property will instead be used to divide the search space into smaller and more easily computable chunks allowing the pareto optimal points between the points of the convex hull to be extracted.

Figure 3.13 shows an overview of the steps of the implemented pareto finding algorithm. In this case five pareto optimal points have been extracted by the convex hull quadrant of interest. This in turn created four search areas between these points wherein pareto optimal points may reside. Any points outside these areas can be discarded, which greatly reduces the number of individuals looked at by the actual pareto front algorithm. Each of the search areas can then be independently evaluated as the real pareto front must pass through the convex hull points. Once the simpler sub-problems have been calculated the results are stitched together to form the final pareto front for the entire population.

Should the number of individuals within each search area be larger than a certain threshold the population is split and the implementation continues in a recursive manner. In case of collinear points the convex hull cannot be calculated and a fallback is used wherein the convex hull is only applied in compatible areas and the remainder calculated directly.

For the two objective case the convex hull complexity of $\mathcal{O}(NlogH)$ means it won't add to the complexity of even the most efficient $\mathcal{O}(NlogN)$ Pareto Front algorithms. Where N is the total population and H is the population describing the convex hull.

Chapter 4

Experiments & Results

This section will describe the experiments done and their results with two primary goals in mind. First the evolutionary framework will need to have its implementation refined to suit the evolution of stereo algorithms, with the aim of giving better and more consistent results for this problem. The framework implementation details will be the focus of Section 4.1, with a later return for further tuning in Section 4.5.

Second the framework will be used in creating new pareto optimal stereo algorithm parameters with the goal of comparing them to the current benchmark results for the selected stereo algorithms. This will be handled in three main parts.

- First a baseline for the algorithm performance will be extracted via the generation of their pareto fronts by use of the created framework. This is the goal of Section 4.2.

- Stereo results can be improved by adding extra pre-processing of the images to reduce noise, as evident in current benchmark results. In this part, a variety of prefiltering options will be explored both to test the flexibility of the framework, and to find a robust filtering strategy to apply to the individuals which will be compared to the official KITTI benchmark results. This will be the focus of Section 4.4.

- In the final phase the lessons learned during these preliminary experiments will be combined and if needed a further experiment will be conducted to improve upon the previous results before a handful of individuals will be selected for submission to the KITTI benchmark for comparison. This experiment is handled in Section 4.6 with final individual selection and KITTI benchmark comparison in Section 4.7.

4.1 Experiments on the Implementation

This section seeks to explore the initial implementation details of the project to get a better grasp on each method and how their individual options affect the performance of the evolutionary search.

Most of these experiments will be done with a quality and a runtime objective to match similar fast stereo algorithm comparisons [72, 73]. However, as the output is not guaranteed to contain a result for every pixel the quality objective must comprise of a weight of both the correctly calculated and the actually marked pixels. Otherwise very sparse, but accurate, or dense, but made-up disparity outputs can be expected to win the evolution. Hence for these experiments the quality objective was picked as the Out percentage as defined in Section 3.3.1.

4.1.1 Parallel Evaluation methods

In this experiment the parallel evaluation methods of Section 3.2.4 will be tested to see if it is possible to significantly decrease the time taken to evaluate each individual without causing unfairness in regards to runtime, due to the more stochastic nature of parallel execution.

For robust results a population of 28 identical individuals is saved to a NSGA-II generation file then reloaded and re-evaluated 100 times to allow runtime statistics to be gathered for further analysis. To reduce compute requirements the experiment was done with only one stereo algorithm. ELAS was picked as it is a potentially CPU heavy algorithm with good potential for parallel computing according to the author [19]. The evaluations were done with on a 4 core Intel I7-870[1] CPU supporting 8 simultaneous threads.

Results

The real-time used per individual and the corresponding speedup-factor (see Equation (3.1)) of the tested methods are shown in Table 4.1. Both parallel methods significantly improve on the serial evaluation times, but the A-method provides a better speedup.

To test the potential unfairness of the methods, the Wilcoxon rank-sum test was applied on the runtime distributions of neighbouring individuals within the generation. Large p-values would indicate that the individuals had related runtime distributions and thereby a degree of fairness to the recorded runtime metric. The neighbouring rank-sum values were tallied and the median value included in the table. The A-method shows a median lack of coherence at the 95% confidence level, whereas the B-method provides similar results to the serial implementation.

Figure 4.1 shows the distribution of the results on the first 8 individuals and the last 4 based on the real-time used from the start of the evaluation of each individual to its end. The individuals 9-24 correlate to the first 8 and are not shown for increased readability. Note that the Parallel A method calculates 8 individuals in unison and that the timing method, at this point, was based on the real-time elapsed from individual start till end. Hence the apparent discrepancy with the median time per individual of the referenced data table.

[1] ark.intel.com/products/41315/Intel-Core-i7-870-Processor-8M-Cache-2_93-GHz

	Serial	Parallel A	Parallel B
Median time per individual	3.07s	0.73s	1.01s
Speedup factor	-	4.20x	3.03x
Ranksum median coherence	P=0.64	P=0.04	P=0.51

Table 4.1: The median time actually spent on each individual evaluation when the total generation time is accounted for. For the A-method this number gives a better view of the total generation runtime than the individual evaluation times noted in Figure 4.1, as 8 individuals are evaluated simultaneously. The speedup-factor(see Equation (3.1)*) shows the runtime gain for the parallel implementations. The coherence factor describes how similar each individual runtime distribution is to the next, and is explained in the text.*

Analysis

Based on the runtime and speedup-factors shown in Table 4.1 a significant improvement can be observed by going with a parallel evaluation approach.

As the parallel tasks are mostly independent of each other these evaluation approaches fall into the embarrassingly parallel bracket which would indicate that a large speedup is possible. In this case a factor limiting the speedup is the fact that only 4 of the threads can be executed simultaneously on the CPU used. The remaining threads are Hyperthreads[2] which are opportunistically switched in when their matched thread is idle or otherwise waiting for a resource. Another limiting factor is the shared memory which will become apparent with the SGBM algorithm in later experiments.

Of the two parallel methods the B-method shows a greater overhead in creating and controlling threads as the lifetime and individual work pools of each thread is much shorter than in the A-method. A larger serial merging step is also required for the method to work. Additionally, as the dataset in this case is not divisible with the number of threads, 4 threads may go idle for each and every individual evaluation. The A-method features this same event at the end of each generation, which given the relative size of the population in regard to the dataset size is a lesser potential computational loss. This would indicate that the performance gap between the two methods is likely to increase when additional individuals are added to the population.

However, the A-method has a significant fairness problem as illustrated both by the coherence rating of the data table and the distributions seen in Figure 4.1. There is a clear staircasing effect in the runtime of individuals 1-8 (and 9-16 & 17-24). In addition an enormous difference on the last 4 individuals of the generation can be observed. The latter is caused by their evaluations having twice the CPU resources available due to the remaining threads going idle after having finished their work. For most calculations

[2] http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html

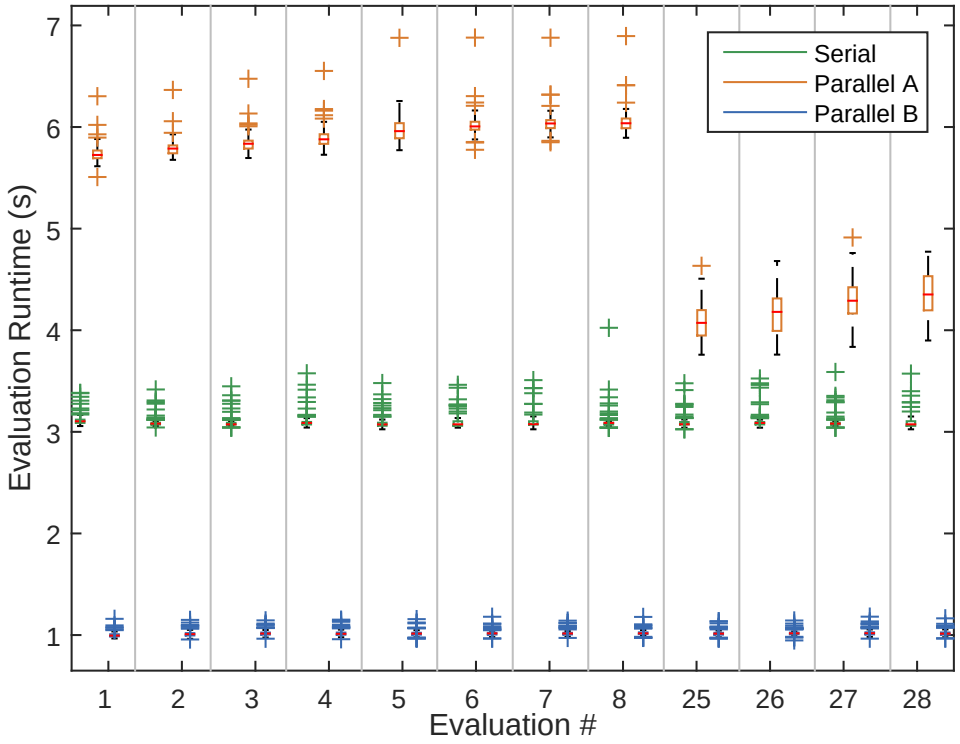


Figure 4.1: Three different computational strategies with evaluations of the same individual over 100 runs. Note that in the case of Parallel A, 8 individuals are evaluated simultaneously causing the actual time taken to be closer to 1/8th of the displayed value. Individuals shown are 1-8 and 25-28, individuals 9-24 not shown due to plot readability, but are like the first group.

this speedup would be beneficial, but in this case it leads to the last few individuals unfairly dominating selection whenever a runtime objective is used. It also skews analysis of found parameters regardless of objectives as the recorded runtimes can not be compared amongst individuals. While this experiment was done with a real-time time-measure it is likely that a similar effect would still be present with CPU time given the variation shown in later experiments. The Parallel B method also suffers from this issue, but in this case it may affect the last image pairs of each and every evaluation levelling the advantage gained. Given the results on Parallel method A, the potential C method was not implemented as it was deemed to suffer from the same unfairness.

A possible solution for the problems with the A-method is to add dummy individuals at the end of each generation causing the last real individual to have a similarly loaded computing environment as the previous ones. However as the order of evaluation is non-deterministic due to the opportunistic nature of the TBB threading library (Section 2.4.3), this would require writing a scheduler and additional code over simply using the inbuilt parallel primitives. This would also have to take into account the wildly different runtimes of each individual so that the last real individual

does not end up computing for longer than the dummies.

Another potential solution would be to re-evaluate the rank 1 front of NSGA-II periodically with a more stable method. This could take inspiration from the implementation of the Online Front Verification method (Section 3.5.2) allowing only new individuals to be subject to this re-evaluation. As limited movement of the front can be expected this method may not require too much in the way of calculation time, and hold potential, but this option will not be investigated further.

Due to the large increase in speedup and similarity of the results to the serial method, the Parallel B method will be the one used in all experiments to follow.

4.1.2 The Effect of Training Set Size

As the size of the training set directly affects both the workload, the number of threads used (see Section 3.2.4), and therefore the total time per run of the parameter search, it is desirable to find a small yet informative training set for further use.

In this section two training set sizes will be tested with the expectation of the larger set producing more general results due to an increase in learning data, while the smaller one was expected to be faster to compute.

Objectives	**Timeout Method**
Out, and Runtime	Hard 6s Real Time[3]
Population	**Generations**
100	480 & 240
Datasets	**Validation**
Training 20 & 40	Best N

Table 4.2: The setup for the training set size experiment.

Due to having the greatest number of parameters and thereby good learning potential, and also having shown itself to have a very wide performance range in the preliminary experiments used to find the initial parameter range limits (see Section 3.4.2), the ELAS algorithm was used exclusively for this test. The further configuration for this experiment can be seen in Table 4.2.

The number of generations was set to give the same number of evaluations on all runs, as such it was set to a number divisible by each of the 40, 30 and 20 sized training sets, even though the 30 set was not used in this version of the experiment. While giving the same number of evaluations, this however may affect the evolutionary pressure as the case with more actual generations will have further selection steps than the evolutionary shorter run.

[3]Accidental choice due to Clock() function, see Section 3.2.5

Results

20 runs were done for each training set size giving the evolutionary progress plot of Figure 4.2. The result for each generation was validated using the Best N method on the Out objective.

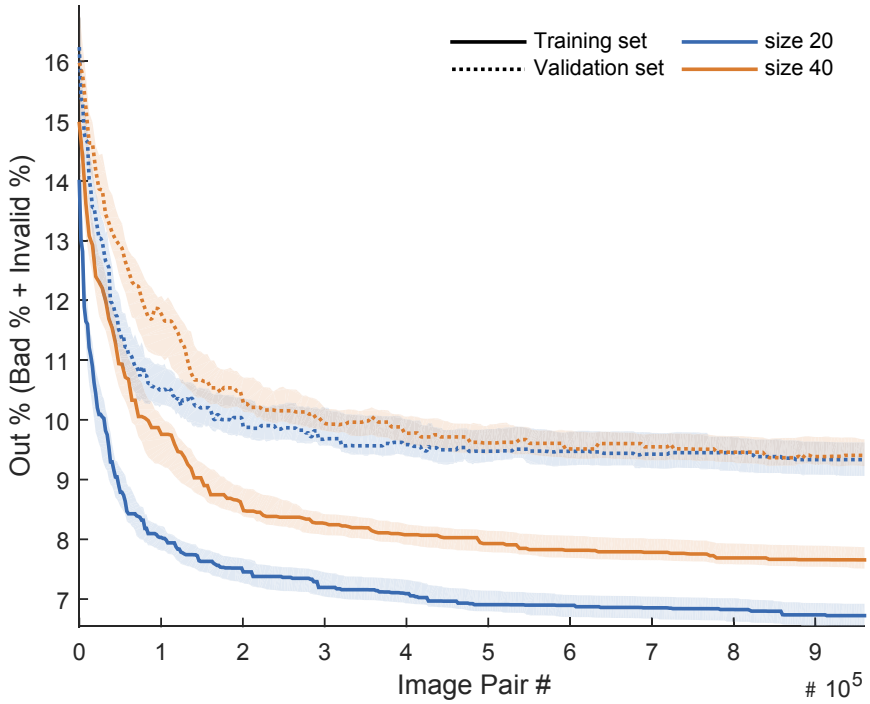


Figure 4.2: The training set size experiment with plotted median best Out percentage and its 99% confidence mean represented by the shaded areas, on both the training sets and the validation set. The X-axis is the generational output normalized to the number of image pairs evaluated.

The mean and median of the final generation Out objective of the runs are listed in Table 4.3 together with the same statistics on the times per evolutionary run. The Evaluation columns of the runtime statistics show the time actually used within the evaluation method (see Section 3.2), whereas the Total columns also includes the NSGA-II and framework overhead.

Set size	Out %				Runtime (s)			
	Training		Validation		Total		Evaluation	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median
20	6.73	6.72	9.34	9.33	82199	80501	80451	79042
40	7.69	7.66	9.45	9.41	65412	63279	63937	62004

Table 4.3: The results from the training set size experiment with the size 20 and 40 training sets as viewed in their last generation. Grey cells indicate the best result.

Analysis

Of the two methods the 40-sized set converges at a slower rate than the 20-set, this is likely caused by the fewer number of generations visited by this method causing less selection to have occurred, but also due to the greater number of unique situations described by the dataset. The two training sets converge to similar validation set distributions at the end of their runs with a p-value of 0.34 between the two. Longer runs may be warranted to decide whether this similarity will continue or if the extra training data of the larger set is able to provide a more robust solution.

Based on the portion of the runtime spent on evaluation, it is clearly slower to compute the 20-sized set than the 40-sized set. This result may seem counter-intuitive, after all, the number of evaluations is constant and the only main difference should be the overhead incurred during selection and offspring production. However, taking into account that the main machine used to generate these results had up to 64 simultaneous threads, it becomes clear that the 40 set simply calculated 40 images simultaneously while the 20 set would need to do the same work in two distinct time-steps, as the maximum number of framework threads is entirely set by the dataset size.

Of the 20 runs, 5 on the smaller training set were done on a different machine and put together they accounted for an overrepresented 1/3rd of the total time. Without these 5 runs the median validation Out becomes 9.17% with a median evaluation time of 70155s for the size 20 set. The loss in quality on these slower runs, is caused by individuals exceeding the timeout threshold and being prematurely stopped. Neither of the machines used in this experiment were exclusively used for this thesis, hence it is likely that on this less powerful machine with fewer idle threads it would be easier for another user to require some of the resources bound to the framework and thereby affecting the timekeeping method currently used. This would indicate that the number of maximum threads and their actual availability can heavily influence both the time taken and the quality of the results. The influence of the timeout methods on quality will be analysed further in Section 4.5.

Based on these observations a change in timekeeping method was subsequently developed so that results could more easily be compared across similarly fast computers. This change to a CPU-time based method aimed at increasing the runtime robustness for machines under heavy computational load while providing similar results independent of thread count. This change is analysed further in Section 4.1.4. Additionally the experiment method of assigning several of the same runs to the same server was later changed to assigning one run of each type to each server, so as to more fairly spread the variance around. This method is described in Section 3.3.6 and is used from Section 4.1.4 onwards.

While no significant difference was found, the minimum work unit of 20 threads of the size 20 set is easier to divide among the variedly sized servers available. Any larger server can simply compute several of these runs at once, hence it became the set used from this point on. The 40 set is

however revisited in the final experiment as found in Section 4.6.

4.1.3 Testing objective functions and helper objectives

According to the research of Knowles [42] and Jensen [36], it may be possible to aid the quality optimisation of a problem by expanding it to multiple correlated objectives. In this experiment certain combinations of the various objective functions of Section 3.3.1 will be tested as objectives with the ELAS algorithm to see if helper objectives better guide the search. This entailed splitting the Out objective into its composing parts, and testing the addition of the maxOut, RMS and avgErr objectives, giving the evolutionary setup as in Table 4.4

Objectives	**Timeout Method**
Out, and Runtime	Hard 3s CPU Time
Bad, Invalid & Runtime	
Out, maxOut & Runtime	
Out, RMS & Runtime	
Out, avgErr & Runtime	
Population	**Generations**
100	200
Dataset	**Validation**
Training 20	Best N

Table 4.4: The setup for the training set size experiment.

Results

A total of 20 runs were done for each method with the output of the validation function stored for each run. The median of these runs and the corresponding standard deviation can be seen in Table 4.5. The distributions of this output were then compared for statistical significance giving the results of Table 4.6. Here the extra correlated objectives of maxOut and RMS clearly reduce the median Out percentage on the validation set at the 95% significance level.

Objectives	Validation Median Out%	Standard Deviation
Out, Runtime	9.946	0.425
Bad, Invalid, Runtime	9.920	0.352
Out, maxOut, Runtime	9.638	0.608
Out, RMS, Runtime	9.773	0.618
Out, avgErr, Runtime	10.059	0.984

Table 4.5: The resulting median Out percentage as output from the validation function and the standard deviation of each method.

Method	Out, Runtime	Bad, Invalid, Runtime	Out, maxOut, Runtime	Out, RMS, Runtime	Out, avgErr, Runtime
Out, Runtime	0.505	0.388	0.957	0.959	0.317
Bad, Invalid, Runtime	0.622	0.505	0.981	0.978	0.289
Out, maxOut, Runtime	0.045	0.021	0.505	0.516	0.066
Out, RMS, Runtime	0.043	0.023	0.495	0.505	0.077
Out, avgErr, Runtime	0.697	0.725	0.939	0.929	0.513

Table 4.6: The p-values on a left-sided Wilcoxon rank-sum test for the tested objective combinations on the validation set. Grey cells indicate that the objective type on the left has a significant median improvement at the 95% confidence level.

Analysis

The maxOut objective put focus on the worst performing image pair of the dataset thereby allowing the more generalising individuals to survive and procreate. This however put much pressure on the results of a single image pair for each individual. As such the RMS objective may be a better choice as it weights in on all images and tries to minimize their RMS per pixel error. However as it calculates only marked pixels it cannot be used on its own.

Splitting the Out objective did not improve upon the previous objective combination. This is likely due to less pressure in the search space region actually tested by the active validation method. The Best N validation method gives a rather limited view of the total population found. Hence it is likely that plotting the entire population based on two or more static objectives would give a better idea of actual objective performance. However the actual evaluation method has so far only output the active objectives making such a post-run analysis impossible. This lead to a change forcing the framework to save all the possible objective metrics for each evaluation to a separate file, so that individuals can later be analysed without concern for the active objectives (See Section 3.5.3). Preliminary testing with this change showed great potential in a multi-objective approach with the entire pareto front validated and displayed. This will be further analysed in the first main experiment of Section 4.2.

The ELAS results interpolation made it difficult to see whether a result was related to excellent algorithm output or just the right amount of output for the interpolation to fill in the remainder. Different algorithm results may hence end up with the same output. As this also precludes looking at density-quality trade-offs, this feature will be limited to its Robotics setting

of 3 pixels in future experiments.

4.1.4 Real-time and CPU-time methods

The experiments of Section 4.1.1 and 4.1.2 highlighted a problem in regard to the timekeeping method used in both potential runtime objectives as well as for the timeout criteria used in the implementation. This problem was tracked to the clock()-function as described in the Timekeeping portion of Section 3.2.5.

The influence of the difference between the real-time (*RT*) and *CPU* timekeeping methods will be examined in this experiment on both an *Idle* and *Busy* system environment. Additionally a runtime objective is used so that the variance of NSGA-II assigned ranks of individuals depending on this non-deterministic objective may be studied.

Objectives	**Runs**
Out, MaxOut & Runtime	200
Population	**Dataset**
20	Training 20

Table 4.7: The setup for the timekeeping experiment.

Table 4.7 shows the setup for the experiment with 200 re-evaluations of the same NSGA-II generation file. Said generation file was the second generation of a randomly started ELAS run. The same machine was used for all experiments in two configurations. The first environment simulates a machine working nearly exclusively with the framework made for this thesis and is made to show the results in a typical single-user environment with no other heavy computational tasks running. This configuration will be referred to as the Idle environment. The second test environment featured another instance of the framework running in parallel to the actual framework tested and with equal operating system priority. This latter Busy approach may better simulate the problems associated with the shared servers used in the other simulations of this thesis, e.g as was seen in Section 4.1.2.

To reduce variance in association with when a particular experiment was run, the variance reduced method of Section 3.3.6 was used to control the machine responsible for all the runs.

Results

The results on this experiment for both the real-time (RT) and CPU time methods are summarized in Table 4.8. The assigned NSGA-II ranks were tracked, and the total number of assigned rank outliers across runs is included in the table. The size of the 99% confidence intervals for the recorded individual runtimes are also included to give an indication to the stability of each method. A tighter confidence interval suggest less

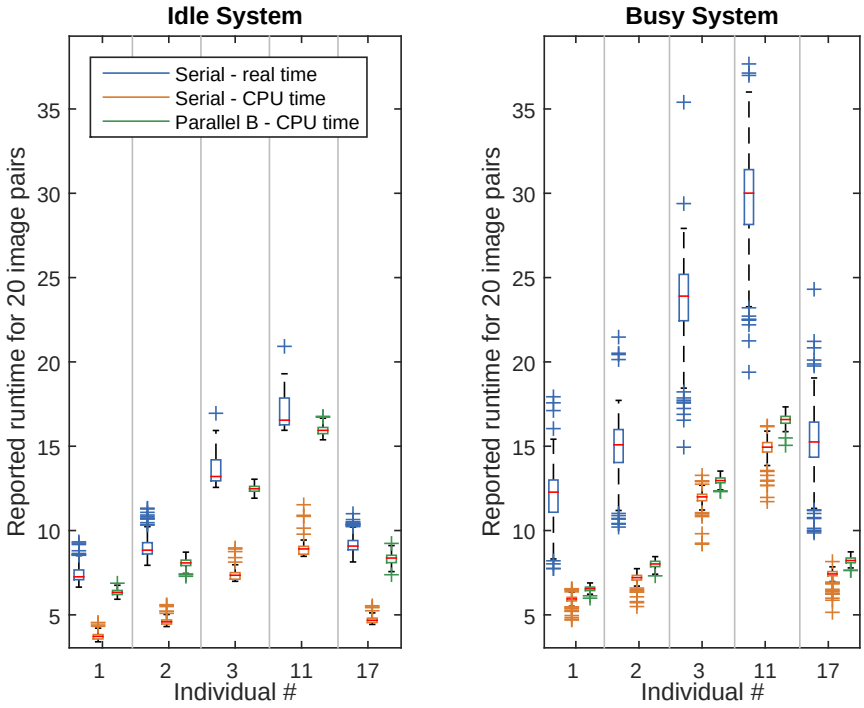


Figure 4.3: A boxplot of the wall clock time (real time) and CPU time with a serial evaluation method as well as the CPU time for the parallel B method. The individuals shown are the first three individuals of the population as well as individual 11 and 17 to better illustrate their behaviour in Figure 4.4.

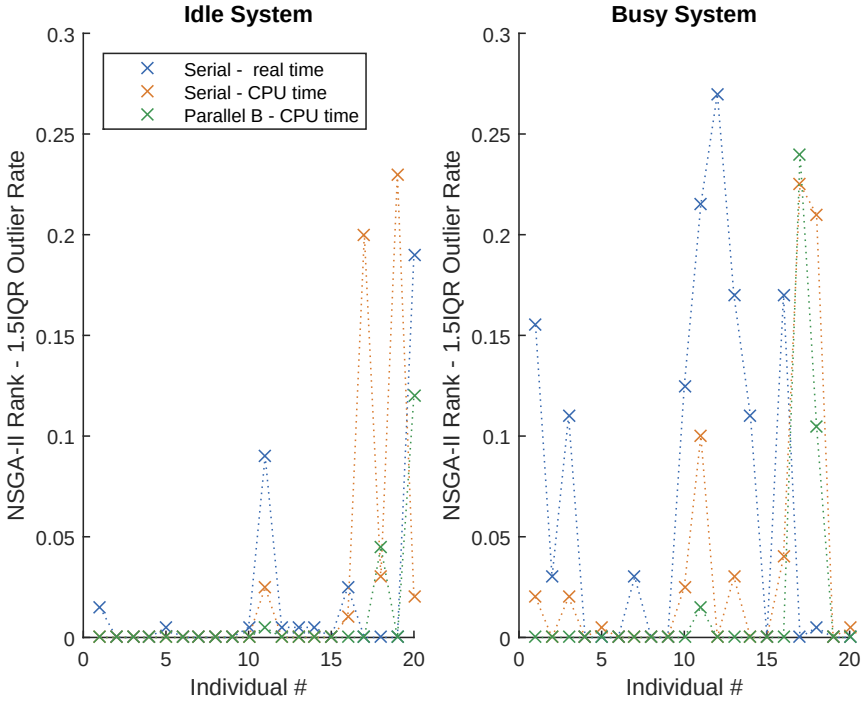


Figure 4.4: The outlier rate for the rank of each individual as assigned by NSGA-II each re-evaluation. Outliers are based outside the 1.5IQR range, that is the area outside of the typical boxplot whiskers.

	Serial - RT		Serial - CPU		Parallel B - CPU	
	Idle	Busy	Idle	Busy	Idle	Busy
1.5IQR rank outliers	69	278	103	136	34	72
Mean confint	0.173	0.638	0.075	0.095	0.054	0.053
Median confint	0.134	0.461	0.049	0.074	0.041	0.039

Table 4.8: This table shows the number of outliers for the different methods in both system environments tested. The 1.5IQR metric is equivalent to the outliers in a typical boxplot (See Section 2.4.1). Also shown is the mean and median 99% confidence intervals for the runtimes recorded for each individual with each method. Grey cells indicate the best results for each of the two test environments.

variation in the distribution. Not all individuals are equally prone to rank outliers, Figure 4.4 provides the portion of outlier runs for each individual. Individuals with near zero outlier rate are situated along the pareto front in areas of little runtime competition, hence allow larger variance before they drop to the next rank. The related plot of Figure 4.3 shows a boxplot of the time reported for a sample of the individuals.

Analysis

The runtime plot of the serial RT method in the idle environment clearly shows the problem with interrupting tasks even on a seemingly idle system. The upper whisker has a much longer tail than the corresponding lower whisker, and all the outliers are towards increased runtime. This can also be seen for the real-time method where the median is positioned low within the plotted boxes, with larger observed variation, yet still very directional outliers.

The opportunistic scheduling of the TBB library in the parallel CPU time method means that even in a busy environment should a thread be significantly delayed others will take some of its non-started tasks when they complete theirs. This in combination with the difficulty of interrupting all the running threads causes the CPU B-method to be more robust than the equivalent serial variant. This is again shown in Table 4.8 with a reduced number of outliers and a tighter distribution in comparison to the other methods.

Based on individual 11 and 17 the rank outlier rates on the busy system give the appearance of these individuals being situated more or less on the decision line between ranks as the CPU time methods have similarly relatively scaled runtimes and both report rank outliers. This led to the development of the Soft Non-Dominated Sorting system of Section 3.3.5 for increased fairness of runtime dependent ranking.

The CPU time function *GetThreadTimes()* generates its result based on counting the number of operating system scheduler CPU time slices used. This leads to system-dependent accuracy in the range of 10-15ms, but with the added caveat that should the time slice not be used in its entirety

the operating system must decide whether to report zero time used or the entire time slice. Added up this can explain some of the outliers present. It is possible that a CPU cycle based timing method could have been used instead, but as this would require a benchmarking of the workers and less reuse of the previous timeout code, it was not implemented, but instead delegated to potential future work (See Section 5.3).

4.1.5 The Uniqueness Check

This experiment will test the Uniqueness Checker of Section 3.2.6, seeing if it is able to uphold diversity while cutting down on potentially unnecessary evaluations allowing for a saving in total evaluation runtime.

This method reacts when the framework is asked to evaluate a set of parameters which has been computed earlier in the same run. It offers three distinct strategies on what to do in case such a repeat should occur, each of which will be tested in this experiment. The Reject and Regenerate strategies will be referred to as the active strategies. The Keep strategy effectively disables the checker.

The Keep, Reject and Regenerate strategies will be treated as seperate test cases with 40 runs of the SGBM algorithm dedicated to each type. SGBM, like the BM algorithm, has few control parameters causing it to be prone to said occasional re-evaluation, potentially wasting effort better used elsewhere.

The Uniqueness checker does not save the evaluated individuals to file, as such the strategy of loading a previously generated partially converged generation was not a possibility as it would require many generations for collisions to occur which at the regular population sizes would require too much computation for a small experiment. As such, an engineered low diversity environment was created by starting with just 16 random individuals and giving them a single objective to optimise, namely the Out percentage. For robust results the 40 runs were done on the same machine. The generation count was set to 60, as the first few generations are unlikely to run into duplicates.

Results

Results for this experiment can be seen in Table 4.9 and Figure 4.5. Plot A and B of the figure shows the Uniqueness checker as a time saving measure which for both the Reject and Regenerate strategies at the 99% confidence level indicates that they are significantly faster than the Keep method. The randomly regenerated individuals are on average faster than the clones they replace.

Plot C shows the median cumulative repeat parameter count and its 99% confidence mean. Both the active strategies show similar development of the cumulative repeat individual counts with a final P-value of 0.75

[4] Of which 11 re-triggered the check after regeneration

	Keep	Reject	Regenerate
Total repeat count	3698	2566	2499[4]
Median repeat count	89	62.5	62.5
Median cumulative time (s)	8296.4	7598.3	8108.4

Table 4.9: The total repeat count across all runs for each strategy as well as the median cumulative repeat count and median cumulative time per run. For reference the total individual count was $16 * 60 * 40 = 38400$ *for each strategy.*

between them. In comparison with the Keep strategy they show independent distributions with a P-value of 0.00. The total repeat counts of each method is displayed in Table 4.9.

As for preserving quality a significant difference could not be claimed at the 95% confidence level for the Out percentage plot of Figure 4.5D. However the results on the Reject box may indicate that dropping genetic material could lead to a larger variance.

Analysis

The Uniqueness Checker in its two active modes was shown to provide a statistically significant reduction in evaluation runtime with the regeneration strategy showing no significant gain or loss in quality. These results and the number of clones in Table 4.9 and its generational trend shown in Figure 4.5C, indicates that the Uniqueness Checker provides a promising addition to future experiments.

The simple single-objective choice was able to provoke parameter duplicates without having a large population size, thereby allowing a larger number of runs to be done to more robustly test the method. However, it also meant that a more thorough analysis of the potential diversity gain was not possible as the single-objective mode of NSGA-II has no reason to spread its solutions making a search space analysis difficult.

The Reject strategy featured the largest reduction in time used, but did not converge as consistently as the other methods. The latter is likely caused by the fact that even a single rejected genome invalidates a large part of the population given the small size used in this experiment. Further runs with a multi-objective context would be required to look into the potential diversity preserving nature of the technique, however this would require a larger population and more evaluation time to complete.

As a fairly small experiment the choice of reduced population size and generation count made sense for the time allocated to the experiment, but a more detailed and significant analysis could have been done if these numbers matched the environment used in the later experiments. Based on the current total simulation time of 11 days this is likely to have expanded the experiment to require 231.5 days. With this in mind this was not looked into in more detail. However the Uniqueness checker trigger count for such a larger environment is briefly revisited in Section 4.2.2.

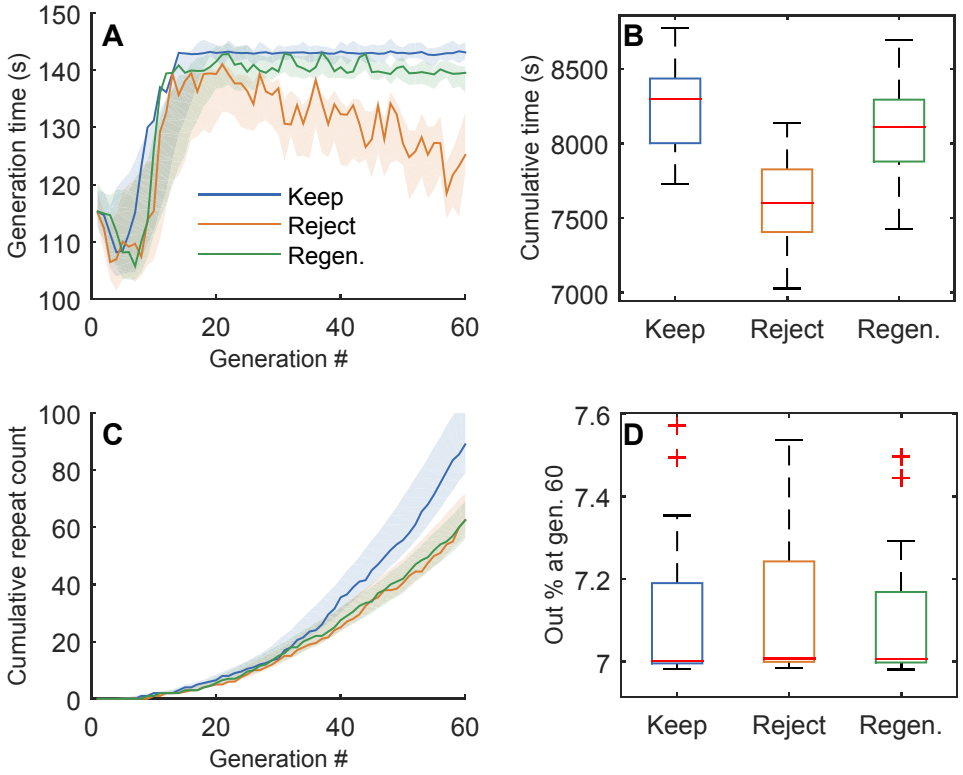


Figure 4.5: The results of the experiments on the Uniqueness Checker. A) Shows the median time per generation with the 99% confidence mean in the shaded area. B) Boxplot of the cumulative runtime per run for each strategy. C) Contains the median cumulative repeat parameter count per generation and its 99% confidence mean. D) Boxplot of the final Out % of the best individual of each run for each strategy.

4.2 Multi-Objective Parameter optimisation

The traditional approach for comparing stereo algorithms is to look at the programs as submitted by their respective authors with that author's chosen parameters. Such results can be seen on the Middlebury[5] and KITTI[6] benchmark pages. However, this method may not sufficiently show the differences in the algorithms as different trade-offs may have been made in regards to quality, density and runtimes. In this section several algorithms will be compared without making these trade-offs in advance. However a runtime constraint is added, both so that the results may have real-time applicability and such that runs are reasonably quick.

This experiment aims to compare the three selected stereo algorithms and see if a multi-objective approach is able to produce more robust results than the current algorithm defaults, and whether insights into each algorithms performance curve can be gathered. The results with the default parameters, as evaluated with the framework computational

[5] http://vision.middlebury.edu/stereo/eval3/

[6] www.cvlibs.net/datasets/kitti/eval_stereo_flow.php

Objectives	**Timeout Method**
Out, RMS and Runtime	Hard 3s CPU Time w/Regeneration
Population	**Generations**
100	200
Dataset	**Validation**
Training 20	Best N and Offline

Table 4.10: The setup for the initial pareto front experiments.

method, are listed in Table 4.11 for easy comparison. Applying the lessons learned during the initial experiments led to the experimental setup as per Table 4.10. Due to the stochastic nature of evolutionary computing, 10 runs were done per algorithm.

The parameter ranges for this experiment were as specified in the *Initial set* of Appendix A, with two changes to the ELAS ranges to account for the results in Section 4.1.3. This entailed locking the result interpolation (inpainting) to 3 pixels as per its KITTI parameter default, thereby making it easier to see the output of the actual algorithm rather than the algorithm and inpainting in conjunction. Additionally enabling the median output filter is now controlled by the evolution rather than being locked to always on.

The results of this experiment will be analysed in the following sections handling each of the different aspects of the results including quality, density, runtimes and parameter distribution.

4.2.1 Quality, Density and Hypervolume

This results section will look at the quality and density of the solutions found by the genetic algorithm. Figure 4.6 shows the Bad vs Invalid parato optimal solutions for each algorithm as generated by merging the total populations of all runs, which allows a more detailed front to be generated. The convex nature of the pareto front and the active quality-preserving and disparity culling measures of each algorithm causes no pareto-optimal individuals beyond 12.5 Bad. For Invalid the generated pareto fronts cover the entire scale, but for ease of visualization and comparison, the lowest density results are not plotted as it makes it harder to decide on a trade-off within the portion more relevant to current benchmark results.

Given the total pareto fronts it is possible to extract new and better algorithm parameter values. Table 4.12 shows some of the values attainable should a fixed density value be desired, as well as two new results improving on only one objective of the default parameters.

A boxplot of the hypervolume for each of the 10 runs as calculated within the objective range of Figure 4.6, can be seen in Figure 4.8. The

[7] Does not include the prefiltering with a 0.7σ gaussian.

[8] Includes full interpolation of the resulting image, hence can not be compared directly. Without interpolation it provides 3.98% Bad and 14.89% Invalid.

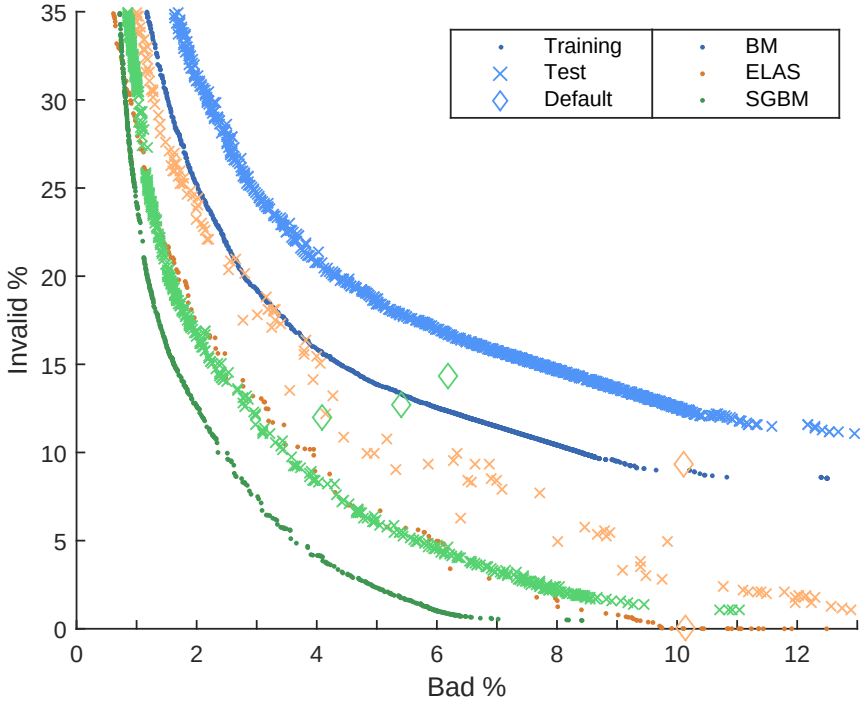


Figure 4.6: The pareto front of Bad % vs Invalid %. Includes both the results on the training set, the test set and the default algorithm parameters as evaluated on the test set.

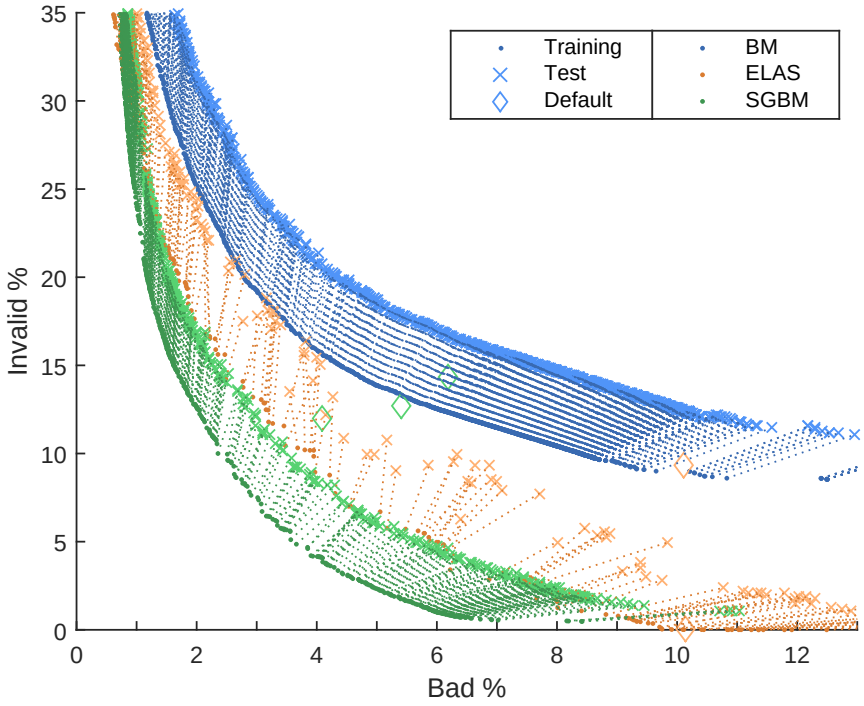


Figure 4.7: The transferability of the pareto front of Bad % vs Invalid % with dotted lines going from each training result to its connected test set result.

BM defaults	Bad %	Invalid %	Time (s)
KITTI A*	6.92	45.60	0.16
KITTI B	1.00	55.32	0.12
Algorithm defaults	1.87	63.11	0.14
OpenCV example	0.74	60.79	0.13
ELAS defaults	Bad %	Invalid %	Time (s)
KITTI defaults[7]*	10.11	9.30	0.64
Middlebury defaults[8]	10.13	0.00	0.75
SGBM defaults	Bad %	Invalid %	Times (s)
OpenCV example	5.40	12.72	3.78
KITTI defaults	6.20	14.30	3.81
Middlebury defaults*	4.09	11.97	4.26

*Table 4.11: The results on the test set with the algorithm default parameters. Includes the parameters used on the official KITTI and Middlebury benchmarks as well as the recommended parameters of an official implementation example. For BM the KITTI parameters are incompletely defined, the A is if the remaining parameters are left as algorithm defaults, B is if the remainder are left as in the OpenCV BM use example. * indicates parameter set used for comparison later on. Time is per image, as calculated within the framework with the same machine as the test set verifications to come.*

wide distribution on the SGBM algorithm is caused by two workers being unable to calculate solutions within the set timeout causing the search to converge to local minimums with respectively 246.37 and 266.17 as their hypervolumes. Without these two runs there is a clear statistical difference between all algorithms at the 99% confidence level and the SGBM distribution tightens up around the current median value. With these runs, a difference can be observed at the 95% level. The ELAS algorithm features a decent variance in its distribution caused by its increased search space complexity as well as the timeout affecting the results on the different worker architectures to varying degrees.

The transferability plot in Figure 4.7 shows how individual results on the training set are mapped to the test set. Individuals specialised in regards to one objective drop significantly in the other objective after evaluating on the test set. Also of note is the regularity of the BM and SGBM algorithms producing consistent training to test movements with much the same distance and objective-space direction. It is possible that such observations can lead to future work using a transferability approach.

Analysis

The results show great promise in selecting a new individual to better represent the selected algorithms on the KITTI benchmark. With the presented front of Figure 4.6, a more informed decision can now be made

	Default	Target Invalid		As Their Best Default	
	(Bad, Invalid)	15%	5%	Similar Bad	Similar Invalid
BM	(6.92, 45.60)	(**7.69**, 15.01)	-	(6.91, **15.83**)	(**0.92**, 45.58)
ELAS	(10.11, 9.30)	(**4.05**, 15.03)	(**8.00**, 4.99)	(9.75, **2.93**)	(**5.87**, 9.29)
SGBM	(4.09, 11.97)	(**2.27**, 15.05)	(**5.62**, 5.00)	(4.02, **8.32**)	(**2.99**, 11.56)

*Table 4.12: Potential quality-density trade-offs. Target Invalid refers to keeping the density constant at the given value then looking for what quality that entails along the pareto front. As Best Default shows the density and quality attainable if respectively Bad or Invalid are locked to the current algorithm default result. The latter comparison is done in regards to the default parameter set as listed in the Default column via the parameters indicated by the * in Table 4.11.*

in regards to whether to focus on moving the current default towards increased density, quality or a trade-off of the two.

Note the uneven distance between points, especially for the ELAS algorithm. A more explicit focus on Bad and Invalid as objectives may have helped in this regard due to the crowding distance having a more direct relationship with this plot. This will be looked at further in Section 4.3 and 4.6.

An interesting observation on the individuals along the front is that they solve different areas of the image with different degrees of success. In case of the sparse individuals in particular there is a high degree of specialization on certain stereo subproblems with some individuals only providing results along certain features while others generalize to larger areas with less accuracy. The specialized individuals return sparse results with very low RMS error around different features, which may lay the foundation for combining multiple results as a possible future work (see Section 5.3). A limited test on just the first image of the dataset was done by combining 32 very sparse and accurate results by weighted sum of the training set RMS error. While not a robust test this was able to halve the average disparity error (avgErr) in comparison to a similarly dense result on that image.

4.2.2 Parameter Distribution

Looking at the parameter distribution along the front may allow gained insight into the function and interconnection of each parameter. In turn this may give an algorithm designer the possibility of improving the algorithm behaviour by reviewing the used values in correlation with the location on the pareto front. Hence, in this section all the used parameter values will be plotted in regards to the objective space position giving the opportunity to see their influence on the front. Additionally, a brief look at the total parameter repeats will also be done.

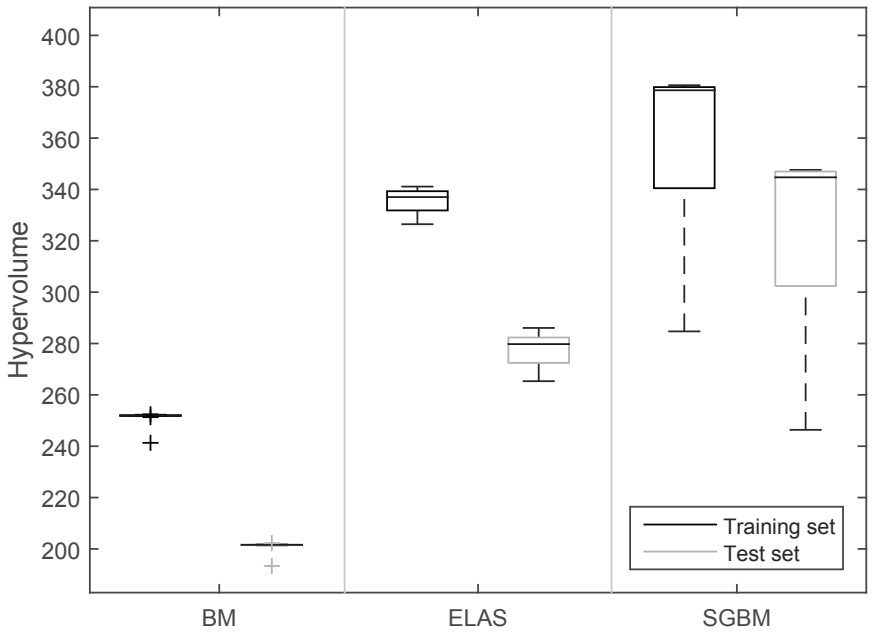


Figure 4.8: Boxplot of the hypervolume of the Bad vs Invalid pareto front for each run. Results shown on both the training and test set. Volume calculated in the same area as Figure 4.6, with the hypervolume reference point placed at Bad 13% and Invalid 35%. For comparison a perfect result would give the maximum possible volume of 455.

Results

Figure 4.9 shows the parameters used for each algorithm along its test set pareto front. Taking all the values on a vertical line returns the active parameters for that Bad percentage. Parameters which show little variance in value tend to be adapted to the image size itself or to its content meaning a very limited range represents the optimal values across the entire front. Examples would include the preFilterType of BM which for 99.2% of the front, was left at the normalization type with a few very sparse individuals left with the Sobel kind enabled. This latter observation would indicate that for this dataset the recommended default BM prefilter type is incorrect as the GA has confirmed that better performance can be achieved with the normalized response filter. For SGBM the mode was always set to full 8-way dynamic programming as the search realised that enabling it had a decent gain with only a runtime penalty it could still fit within the set timeout.

It can be noted that some parameter values specialize in certain parts of the pareto front, of special note are the various quality-preserving measures built into the algorithms e.g speckle removal, left-right thresholds and texture requirements. Their values show a clear indication of the trade-off inherent to enabling these methods allowing more certain matches to be preserved at a cost of density.

Other parameters show a high degree of variance even within roughly the same portions of the pareto front. This shows not only the difficulty

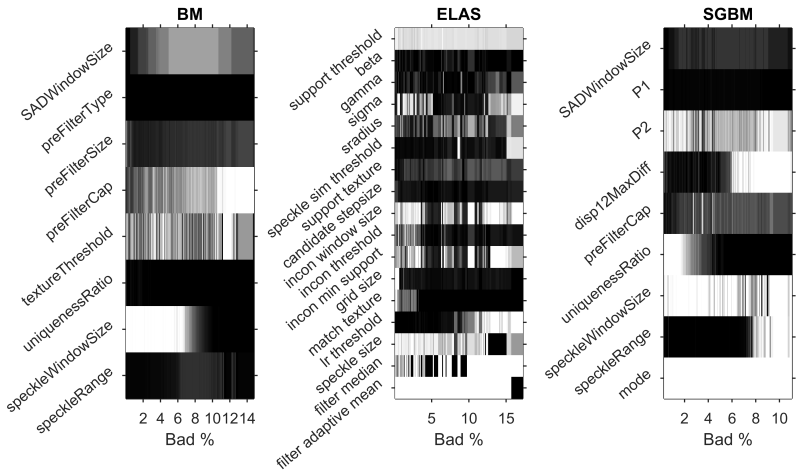


Figure 4.9: Parameter analysis of the test set pareto front of Figure 4.6. The parameter values of individuals along the front are represented using a greyscale indicator on where each parameter is in its allowed range. Hence a white value represents the top of the range of that parameter, while a black value indicates the bottom value with shades of grey between the two extremes.

in parameter optimisation, but also goes to show that several parameter combinations can lead to similar quality output. However, the different parameters could end up calculating in other portions of the image which would explain the varying behaviour observed in Section 4.2.1. The intermixed ELAS parameters and their spread would also explain the reasoning behind the large and varied movement on the transferability plot of Figure 4.7 with, in certain cases, nearly identical training set results mapping to wildly different test set results.

	BM	ELAS	SGBM
Median Uniqueness Trigger Count	177.5	0	35
Uniqueness Trigger Total	1982	0	335

Table 4.13: The number of parameter repeats in all the runs of this experiment. The median is per run.

Table 4.13 shows the parameter repeats across all runs. No collisions were reported for the large search space of the ELAS algorithm, while the SGBM algorithm had some repeating parameters and the BM quite a few.

Analysis

Little total repeat parameters are encountered based on the activity of the Uniqueness Checker module in this experiment when compared to the results in the smaller sized original experiment of Section 4.1.5. It is likely that the additional objective is better able to spread the solutions in the search space leading to fewer parameter collisions. However, the checker is

also not well tuned for real-valued parameters, as the likeness threshold is not scaled with the allowed range for each parameter.

The parameter distribution along the front was analysed to see if the current parameter ranges were too restrictive for the search. For the BM algorithm the *textureThreshold*, *speckleWindow* and *speckleRange* parameters were found to be potentially too restricted as a significant portion of the used parameters were at the top end of the available range. The *preFilterCap* was left as is due to algorithm restrictions on the allowed range. For ELAS the *incon_window_size*, *incon_min_support* and *speckle_size* parameters were certainly too restrictive, while a further 5 parameters showed some potential for expanded ranges and were modified accordingly. The potentially over-restricted parameters for SGBM were *P2*, *disp12MaxDiff*, *uniquenessRatio*, *speckleWindowSize* and *speckleRange*.

This led to the construction of an extended parameter range set as can be seen in Appendix A. On the other hand, underutilized parameter ranges were left as they were as the extra range may be come into play when combined with the behaviour gained from the other newly extended parameters. This updated range will be used in the experiment of Section 4.6.

4.2.3 Runtime Results

This section plots the runtimes of each individual along the pareto front of Figure 4.6. The runtime results as calculated on both the training and test set and normalized per image are presented in Figure 4.10.

Analysis

To better explain the clear clustering in the runtimes of Figure 4.10, a parameter analysis was done by plotting each individual parameter against the runtime as in the example given in Figure 4.11. The clear grouping seen for the BM algorithm is primarily caused by the prefilterCap parameter going above 31, causing the algorithm to switch to an alternate and slower calculation mode. Otherwise the runtimes are largely invariant of the parameters chosen, which is in tune with the claims of the author [43]. For the SGBM algorithm the two primary groups on the training set are not correlated to the parameters used, but is instead caused by the different CPU architectures used for the simulations, as can be clearly seen in the test set results done on a single machine. The clustered fast results on ELAS are caused by several parameters reducing the number of support points to points of very high confidence only. This in turn reduces the calculations needed, but provides very sparse results.

Based on the 3 second runtime roof many of the individuals along the total pareto front will be too slow to be considered near real-time if the criteria of 1 second from Tippets [72] is to be used. Section 4.2.5 takes a second look at the found individuals by focusing on the faster solutions, while accounting for the time bias caused by the parallel evaluation method.

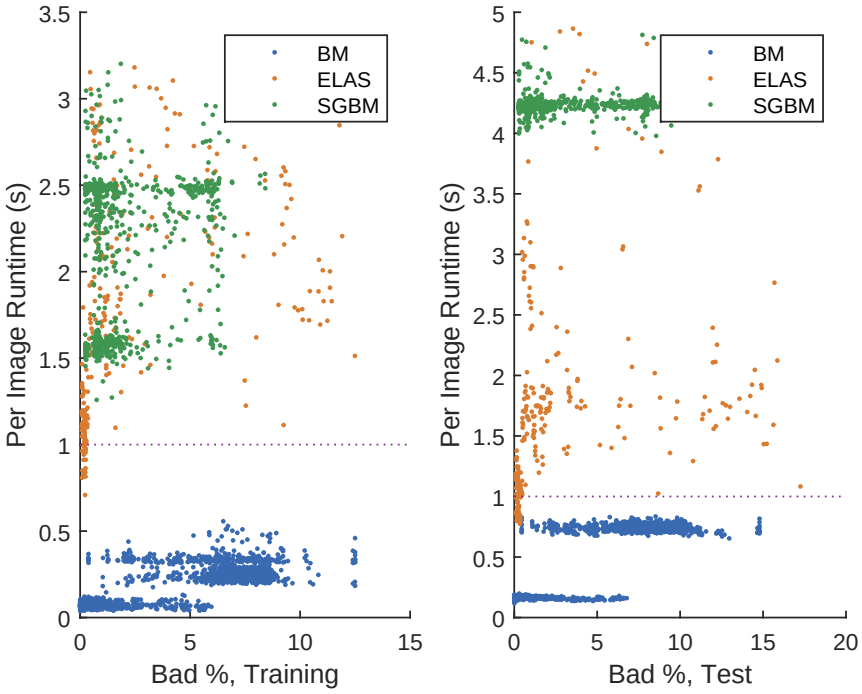


Figure 4.10: The runtimes of each individual of Figure 4.6 as output from the multitude of worker machines on the training set and the output on the test set using a single machine. The algorithms heavily reliant on specifically optimised CPU instructions (BM and SGBM) were slower on the test set due to a difference in CPU architecture. The dotted lines mark near real-time.

Due to individuals adapting to the three second timeout and the varying performance of the simulation workers it is likely that many otherwise good individuals were thrown out on one worker while accepted by others. A possible future work in this regard is to benchmark performance in advance then using a scaling factor based on that benchmark during the evaluations. This is relegated to future work and discussed in Section 5.3.

4.2.4 Comparison with single-objective optimisation

In this section, data from the current main experiment will be used to compare this multi-objective search strategy to a single-objective approach by looking at the total generated populations in the Bad-Invalid objective space. To be able to make the comparison 10 new runs using a single objective were done utilising the same setup as in Table 4.10. The objective for these runs was set to the Out percentage, as optimising simply the Invalid or Bad metrics would inevitably lead to respectively guesswork and no answers given.

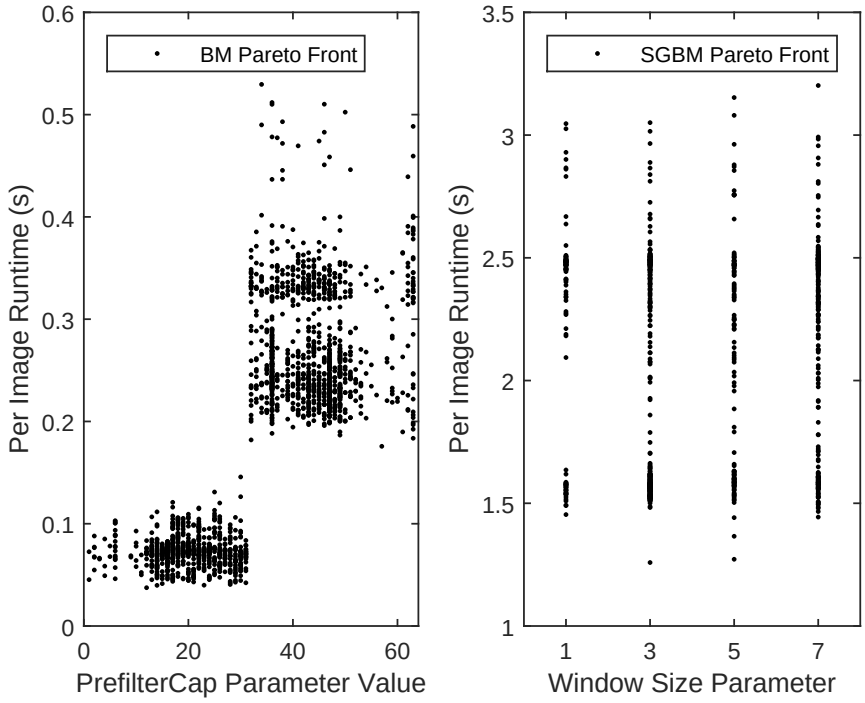


Figure 4.11: A sample of the parameter analysis done on the Bad vs Invalid fronts of Figure 4.6. The left plot shows the clear distinction in runtime happening as the BM algorithm switches calculation mode. On the right the window size invariant behaviour of the SGBM algorithm can be clearly seen. Note that both plots are based on the output on the training set which causes some extra runtime variance due to the different workers used.

Results

Figure 4.12 visualizes density in the Bad-Invalid objective space with all individuals across all runs filled into a 3D histogram. The single-objective method converges well, but tends to search for solutions in a very limited area making it difficult to discover potentially better trade-offs than the exact currently set objective. The multi-objective approach features a good, if somewhat uneven spread across the objective space. As the search objectives are not all directly correlated with the plot axis there are certain density peaks not on the final pareto front. Of the two density peaks at very high Bad percentage, the top left one is certainly caused by the runtime objective as the very lowest runtimes are represented by individuals in this area. For the second peak the distinction is not as clear cut, but it is likely that during the search a significant effort is used in this corner trying to find the global minimum Invalid, even though the presumed optimum is closer to the left as in the pareto front plots of Figure 4.6. Newly generated individuals may also end up with similar density, but worse accuracy than their parent leading to bunching up at the edge of the objective space for particularily unlucky individuals.

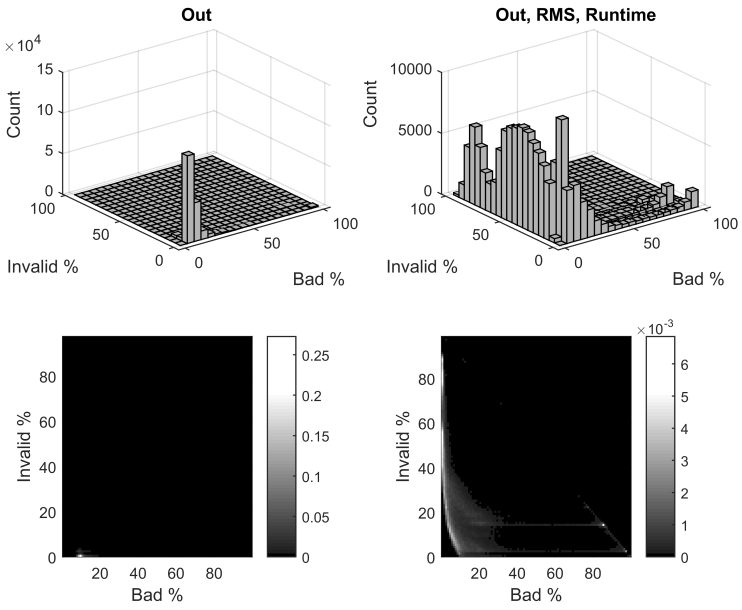


Figure 4.12: Histogram comparison of the search space of the current multi-objective approach and the simple single-objective approach. The top two plots show the 3D histogram of the Bad-Invalid objective space with the individual count in each histogram bin shown on the z-axis. The bottom two plots are the flattened versions of the top two with a greyscale value assigned according to the population density colour-bar along the right hand side of each plot.

This latter tendency also explains the distinct line going from the bottom right to the top left as it represents any individual wherein all output pixels are either marked as wrong or uncalculated. This causes the individual to hit the edge of the objective space as Bad+Invalid can not be larger than 100% in total. The lines going left towards higher accuracy from the previously mentioned two peaks feature some of the same qualities as the peaks themselves hence are also likely caused by recombination.

The RMS objective helps to preserve and generate solutions of all densities with highly accurate, if sparse, results. The lack of solutions along the Bad side of the plot is primarily caused by the quality preserving measures built into the algorithm, but also assisted by the lack of an explicit density objective.

Analysis

It is quite clear that using a single-objective with scalarisation will require multiple runs with different objective weights for a decent representation of the potential trade-offs the algorithms are capable of presenting.

Using NSGA-II with 1 objective means that population diversity is not enforced as unless duplicate objectives are found each front will contain 1 individual meaning that the crowding distance metric is effectively

disabled. This leads to a very high degree of elitism with the top N individuals selected.

The sparsity of individuals along the multi-objective front may lead to poor parent selection, as parents will be specialized to quite different problems. A larger population would be required to compensate for this effect, which is a lesson that will be applied in the last experiment (Section 4.6). Also, the somewhat uneven distribution along the front of the multi-objective approach reinforces the impression gained in Section 4.2.1, that a change in objectives may be in order to better capture the pareto optimal front. However, an even distribution would also require an increase in population as otherwise the evolutionary pressure would be too low in the desired region of interest causing the search to stick to a local minimum.

4.2.5 Results on Near Real-time Individuals

As mentioned in the look at front runtimes in Section 4.2.3, it is likely that the allowed extra room outside the near real-time threshold of 1 second would push the runtimes towards this threshold. In turn this focused the testing on the flexibility of the algorithms to fill that timeout rather than create the best near real-time solutions. In this section the data of Section 4.2 will be given a brief second look by removing all results outside of the desired runtime threshold of 1s. Given the difference in runtime on the default parameters as evaluated with the parallel strategy through the framework and the listed runtimes on the official benchmark, it is clear that the algorithms scale differently and that a simple 1s threshold can not be applied to select the individuals that are likely to produce near real-time results outside of the framework.

Additionally, based on the behaviour seen in the runtime plot of Figure 4.10 simply applying a larger threshold is unlikely to be able to capture all the candidate individuals in a fair manner. This is due to the variation in runtime observed on the different workers. As such the following rules for picking likely fast algorithm parameter candidates will be applied:

- *BM* individuals on the current combined front are all fast enough as is. Result from the main experiment can be compared directly.
- *ELAS* appears to be about half as fast when 20 instances are evaluated in parallel on the test set machine. A threshold of 2s may then select good candidates, but to give a margin from results from the fastest workers the threshold was set to 1.5s.
- *SGBM* is heavily affected by the framework, but happily presents only two distinct runtime groupings based on the optimisation *mode* parameter. The official KITTI benchmark indicates that the 8-way optimisation mode was close to the near real-time threshold. This experiment will test the faster 5-way mode by extracting all the individuals with this parameter setting.

As the new candidates for the ELAS and SGBM algorithms are unlikely to be present in the previously calculated pareto fronts, the previous test data can not be used as only a selection of the total individuals were previously verified. The new individuals were extracted from the total run population and new fronts were generated based on them.

Results

Figure 4.13 shows the combined pareto fronts after thresholding the population based on the experiment rules above. Good results are still generated, but less improvement is made over the default parameters. The ELAS and SGBM algorithms both show a drop in overall hypervolume with a particularily marked worsening for the SGBM algorithm. Their pareto front crossover point moves a significant distance nearly doubling the range of which the ELAS algorithm is the overall pareto optimal choice. A boxplot of the runtimes of the new pareto front is presented in Figure 4.14.

BM	All	Fast only	Change
Median runtime	0.71	0.71	-
Hypervolume	203.9	203.9	-
ELAS	All	Fast only	Change
Median runtime	1.99	1.34	0.65
Hypervolume	293.0	288.1	4.9
SGBM	All	Fast only	Change
Median runtime	4.16	3.82	0.33
Hypervolume	349.5	323.7	25.7

Table 4.14: Comparison of the newly generated combined near real-time front as compared to the combined front of Section 4.2.1.

The results are summarized in Table 4.14 with a comparison to the earlier generated front. The All-column refers to the earlier generated front. Fast only refers to the new faster results.

Analysis

Should the assumption hold of the noted ELAS runtime being half as fast as an instance calculating one image at a time, it is likely that all the individuals on the new front are sub 1s solutions. The small loss in hypervolume and the combined pareto front plot both indicate that faster individuals are able to produce similarly good results via different parameter combinations. After selecting what was believed to be only sub 1.5s individuals for the ELAS algorithm it is clear by the upper tail of the runtime boxplot that parts of the current front came from more powerful machines, as these solutions are now slower than the selection threshold.

Locking the SGBM mode variable to the 5-way dynamic programming method constitutes a larger algorithmic change and the results change by a large margin. The results differ less when very dense solutions are desired,

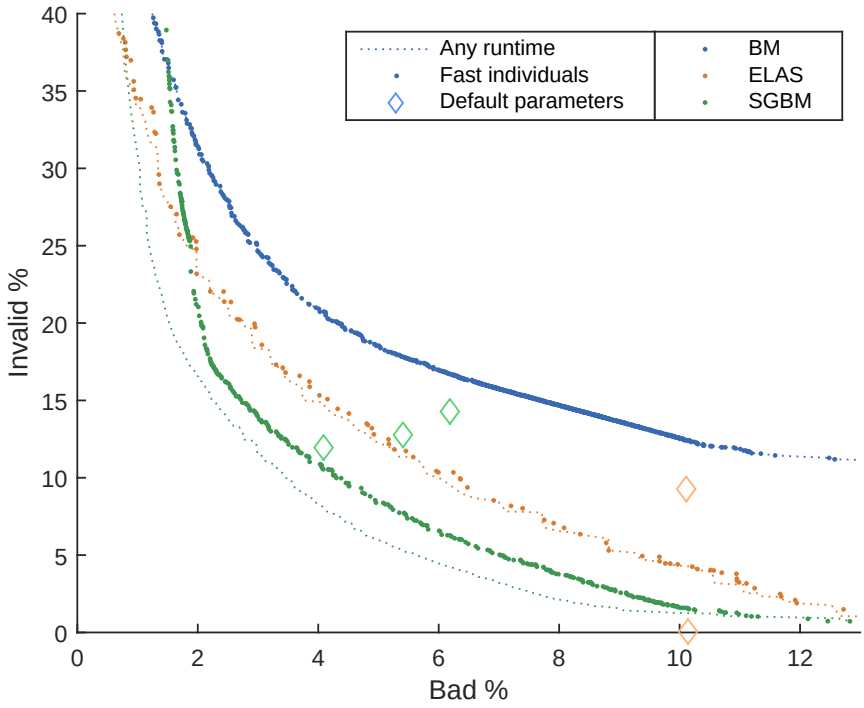


Figure 4.13: Objective space plot of the sub 1 second per image individuals and their combined pareto front as calculated on the test set. Also shown is the original combined front and the algorithm default parameters.

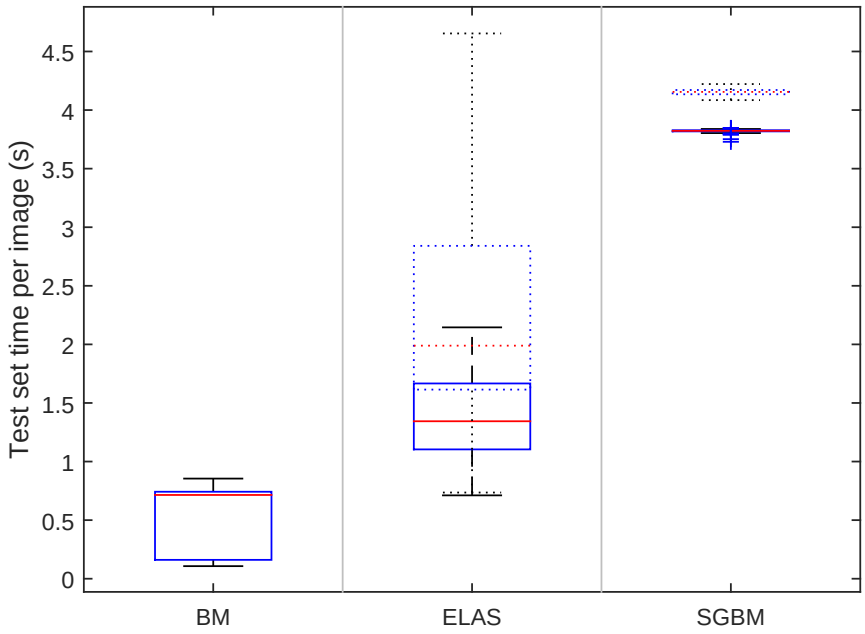


Figure 4.14: Boxplot of the runtimes on the testset of the sub 1s solutions extracted from the training set data. Dotted boxes correspond to the original slower combined pareto front as plotted in Figure 4.13.

with the gap widening as quality improves. This matches well with the description of this control parameter as it is known to increase quality at the cost of time and memory requirements. However it should be observed that even this new rapid set of solutions dominate the 8-way mode default parameters.

Without a compensating step for individual worker performance extracting the rapid individuals with precision is difficult when the solutions had adapted to local hardware performance. A possible solution is a worker benchmark and compensation step, which will be discussed in Future Work (Section 5.3).

4.3 A Brief Look at Overfitting and Generational Progress

A potential problem with learning algorithms is the possibility of overfitting to the training data causing tests on independent datasets to get very poor results. If each pixel of the training dataset is viewed as a unique sample, the training data is enormous allowing good generality of found solutions. Simultaneously the number of different scenes remains fairly low giving the potential for overfitting to the scenes present. The training set size experiment (Section 4.1.2) touched on this subject with its generational single-objective verification, but as the hypervolume metric was not available at that time it may not have given a detailed enough verification. While the runs on the test set so far have shown good results a look at the hypervolume each generation may provide a better view than the current end of run or single-objective approaches used so far.

The experiment setup will be as in Section 4.2 with the exception of a change in objectives to Bad and Invalid to match the hypervolume plotted. With the hypervolume plotted each generation, and the limited population, it was expected that any other objectives would give a larger generational variance as they are merely correlated to the plotted hypervolume. For verification the Front validation method is used with the smaller Validation set (Appendix B.2.2).

4.3.1 Results

A total of 6 runs were done with the front validation active. The generational hypervolume on both training and validation sets can be seen in Figure 4.15. NSGA-II will spread its solutions along the entire pareto front, as such limiting the hypervolume to the range of interest of the previous experiments will not indicate the full state of the population for that generation. Hence, for this experiment the hypervolume is calculated with its reference point at (100, 100), as it represents the maximum possible value for the selected objectives.

An example of the generational progress of one of the runs was

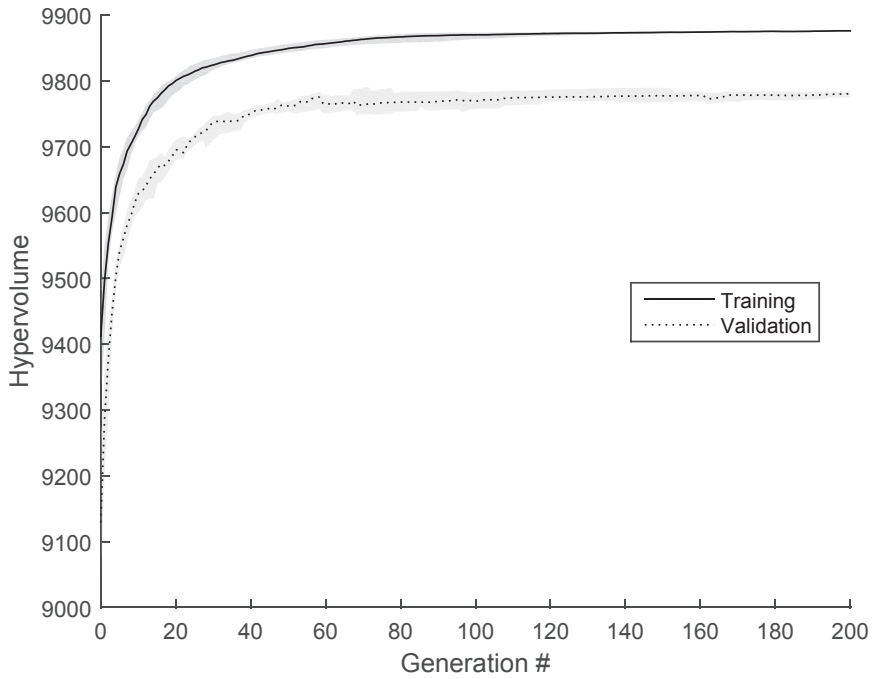


Figure 4.15: The Bad-Invalid hypervolume in training per generation and its corresponding hypervolume on an independent validation dataset. The shaded area represents the 99% confidence mean.

recorded and uploaded to Youtube[9]. This video shows the pareto front and hypervolume on both training and test for each generation.

ELAS	Previous Baseline	This Experiment
Median Pareto Front Population	152.5	285
Combined Pareto Front Population	260	354
Combined Hypervolume (training)	345.6	351.6
Total Individuals Evaluated	200000	120000

Table 4.15: Comparing the size of the generated pareto fronts after a change in control objectives. The Baseline is the resulting individual counts for the Section 4.2 experiment. This experiment lists the new resulting individual counts.

The change in objectives allowed the pareto front to be better defined with more individuals spread along the front even though the total examined population was reduced. The resulting statistics on the sizes of the pareto fronts are listed in Table 4.15. As verification was done on a different dataset, the total training hypervolume is also listed for comparison.

[9]https://youtu.be/__sy1wsUvHw

4.3.2 Analysis

As in the training set size experiment of Section 4.1.2 a similar training to validation distance can now be seen with the more robust hypervolume metric. There is no great loss in validation output over time indicating no noticable overfitting.

The increased confidence interval magnitude starting around generation 70 appears to coincide with the point where the size of the first rank overflows the maximum, and a pruning operation of the pareto front has to take place via the crowding distance metric. This may slightly reduce the hypervolume as indicated by Figure 3.12.

Close examination of the training hypervolume shows this occasional reduction as well. Figure 4.16 shows the effect with its top plot showing run 1 and its rank 1 population count with generations marked by whether a rise or a drop in hypervolume occurs. The bottom plot shows the portion of runs featuring this effect for each of the generations and a trendline indicating the potential future result. Generation 200 is nearly always marked as a loss due to the inner working of the Front validation method. It validates the front before survivor selection, hence in the case of no generated offspring as in generation 200 a loss is generally observed from the more populous mixed parent offspring population of the previous generation. While it says nothing in regard to the size of the hypervolume movement, based on the trendline of the bottom plot the search progress is expected to crawl to a halt sometime after generation 279 wherein an equal number of runs will feature a drop as an increase. This in combination with the rapid filling of rank 1 would strongly indicate that a significantly larger population count would lead to a better search, as NSGA-II would have the room for its rank strategy to work.

The somewhat sparsely defined front observed for the ELAS algorithm in Section 4.2.1 is greatly assisted by the change in objectives. However, less pressure is directed towards minimizing the total Out% causing an occasional loss in that area of the pareto front. This is helped by combining the fronts from multiple runs giving a better defined front overall than the previous baseline. It is likely that an increase in population size would make for more robust results per run. This objective change will hence be used in the later experiments of Section 4.5 and 4.6.

4.4 Results With Prefiltering

When doing changes to an algorithm it can be difficult to fully grasp how the alteration affects the performance of the algorithm. In this experiment such a change will be applied to the three algorithms to see if the multi-objective evolutionary approach can assist in highlighting the objective-space difference of the change. Additionally this experiment serves the dual purpose of applying additional prefilters to the algorithms so as to find a good combination for later experiments and publication to the KITTI benchmark. As per the current results on the KITTI benchmark, where

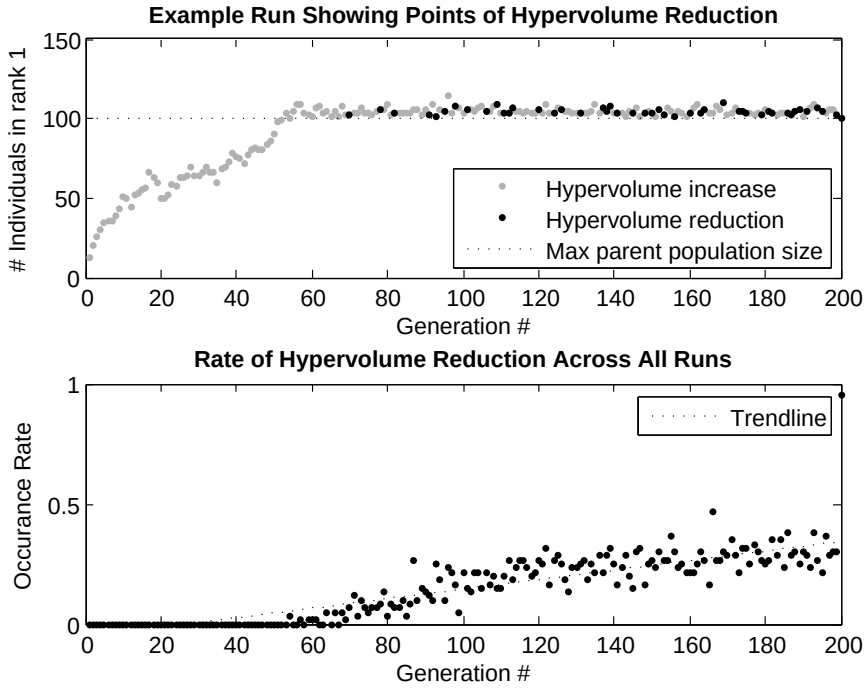


Figure 4.16: Analysis of loss in training hypervolume. The bottom plot is an analysis of 12 runs using the same evolutionary setup without the time consuming generational pareto front verification.

several candidates apply noise reduction to the input images, it is expected that this step will provide more accurate results.

For comparison with the Baseline experiment of Section 4.2 the evolutionary setup remains as in Table 4.10. The prefilters themselves are controlled by additional parameters tacked onto the genome of each individual. As adding prefilters is likely to affect the optimal algorithm parameters, both the prefilter and the filter parameters remain controlled by the genetic algorithm.

4.4.1 Results

Table 4.16 shows the number of runs for each prefilter strategy and the associated time per run. The MeanLoG variant was not yet implemented for the first few runs hence it has fewer runs in total. The reduced number of runs on the SGBM algorithm is due to their long simulation times and losing access to the machines used for the simulations. The latter also caused the uneven number of runs for SGBM as each worker had been started at different times and were at different points in their 9 filter work queues when they were prematurely stopped. Simulations on this experiment amounted to approximately 245.9 days worth of machine real-time, but this was spread across many workers to allow results to be gathered more rapidly. The workers within the same algorithm spread their work across filters to reduce runtime variance. The various algorithms were in part run on different workers.

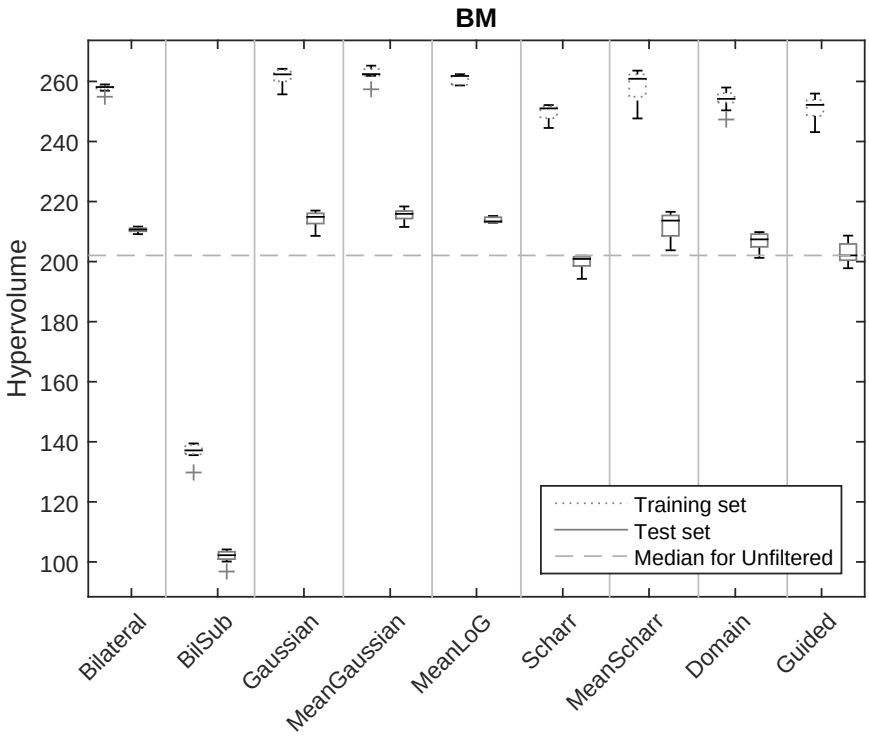


Figure 4.17: The hypervolume for BM on each of the prefiltering strategies.

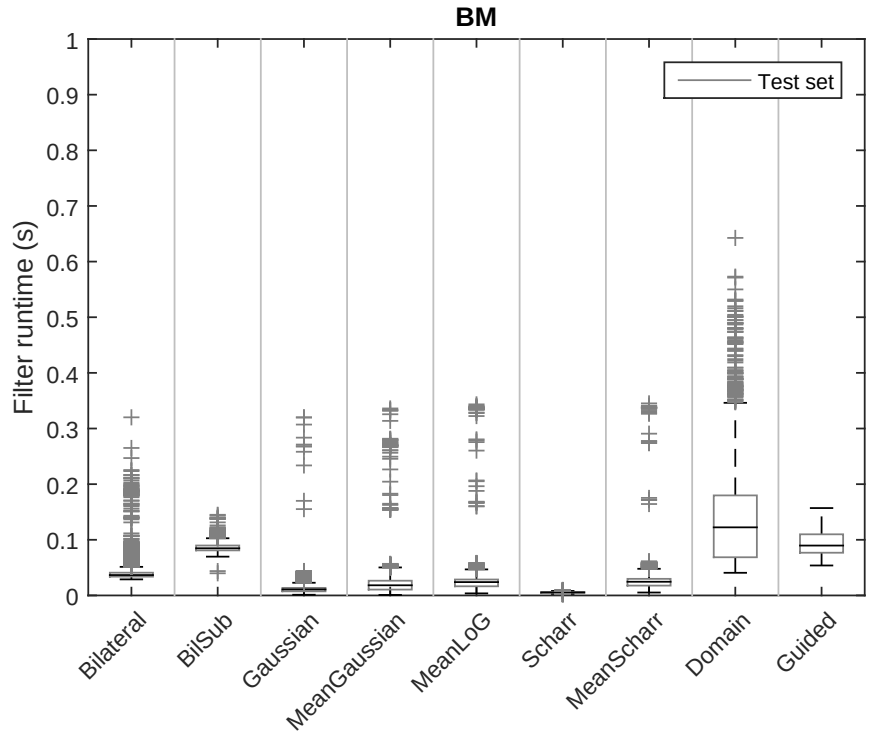


Figure 4.18: The time used on the actual prefiltering per BM individual.

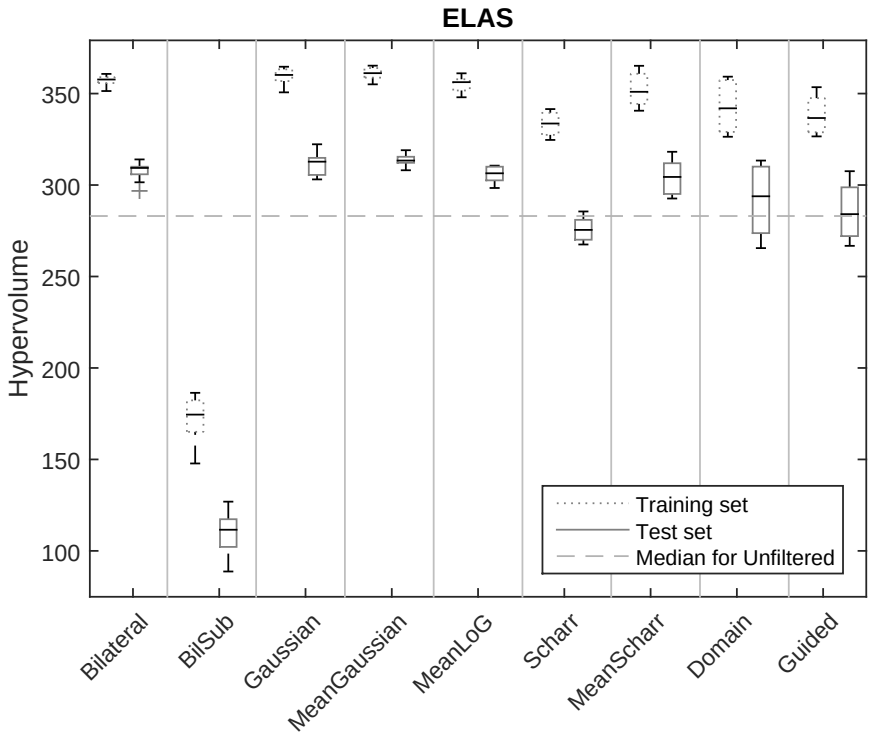


Figure 4.19: The found hypervolume for ELAS with each of the prefiltering strategies.

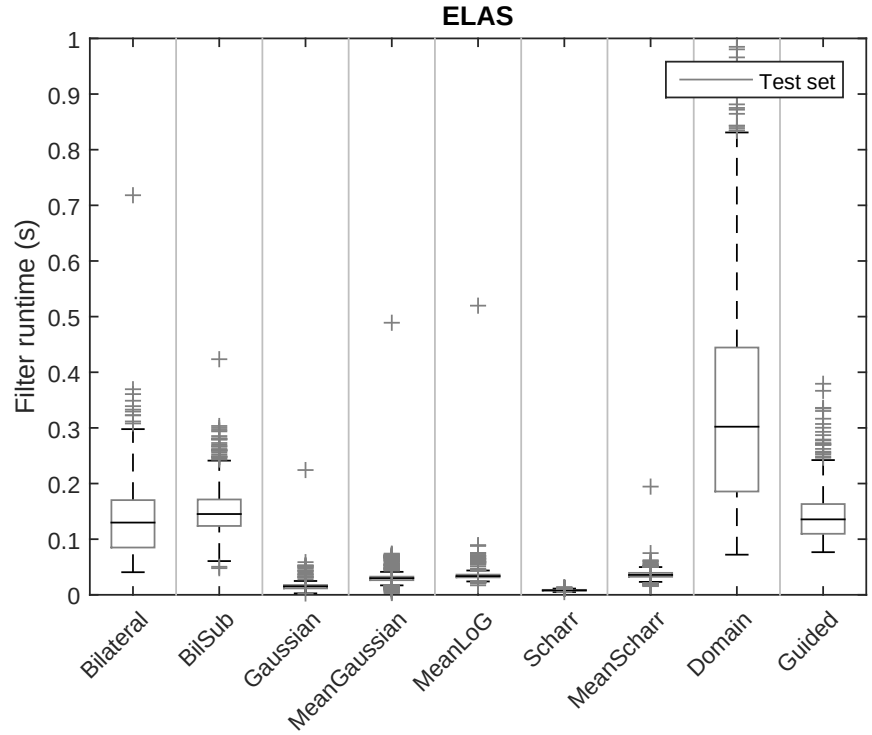


Figure 4.20: The time used on the actual prefiltering per ELAS individual.

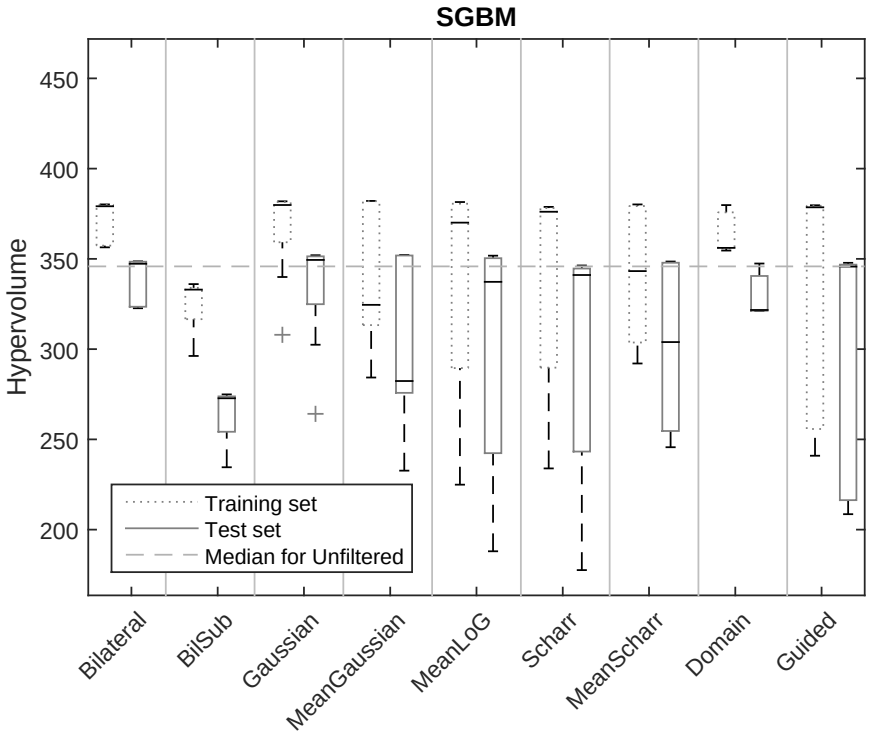


Figure 4.21: The hypervolume for SGBM on each of the prefiltering strategies.

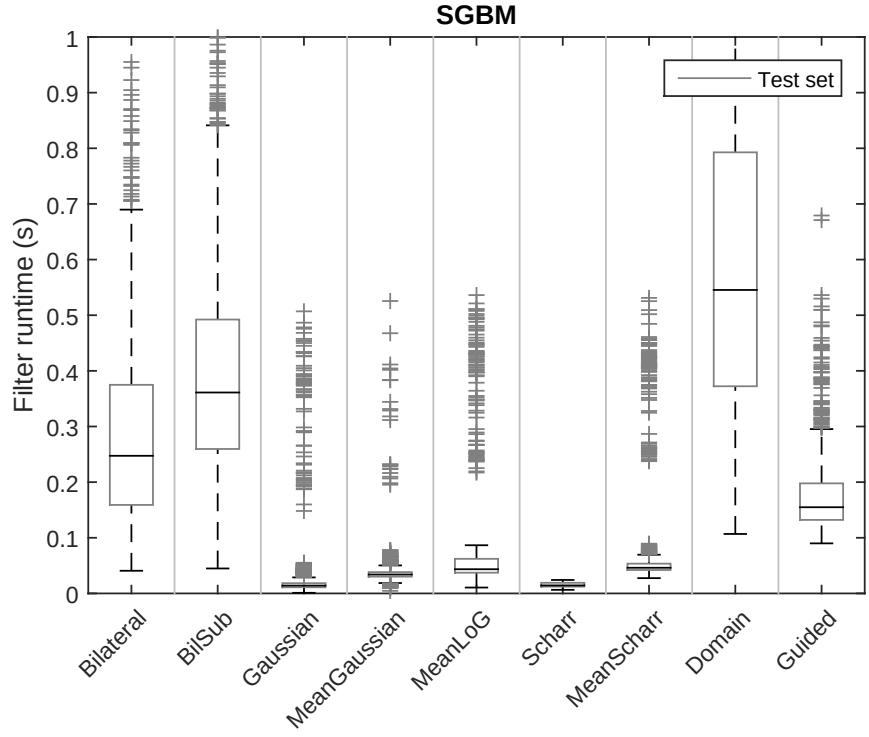


Figure 4.22: The time used on the actual prefiltering per SGBM individual. The Bilateral, BilSub and Domain filters are made slower in comparison to the BM results due to resource conflicts with SGBM executions running in parallel.

Prefilter	Runs			Median hours per run		
	BM	ELAS	SGBM	BM	ELAS	SGBM
Bilateral	9	10	5	2.98	18.96	35.99
BilSub	9	10	8	2.86	17.67	36.18
Gaussian	9	10	8	2.06	17.25	42.76
MeanGaussian	8	10	6	2.67	16.05	50.27
MeanLoG	5	8	7	2.47	16.04	37.64
Scharr	9	10	7	2.42	19.10	38.70
MeanScharr	9	10	8	2.41	21.63	44.30
Domain	9	10	6	4.47	17.40	40.40
Guided	9	10	6	5.55	22.88	56.55
Verification of	Front Individuals			Hours to verify		
BM	24250			24.4		
ELAS	11094			37.0		
SGBM	11252			82.7		

Table 4.16: The number of runs done for each experiment and the median hours used per run for each of the prefilter type used in this experiment. The Verification portion shows the number of total individuals summed up from each of the Bad-Invalid pareto fronts, and the time used to verify them all on the test set using the Offline method.

The hypervolume of the output of each run on all the prefiltering strategies was calculated based on the same region of interest as in Section 4.2 with the reference point at (13,35) in the Bad-Invalid space. This resulting hypervolume can be seen in Figure 4.17, 4.19 and 4.21 for the BM, ELAS and SGBM algorithms respectively. Matching plots for the time used on the filters along the same pareto fronts are included in Figure 4.18, 4.20 and 4.22.

The ELAS Gaussian is not fully able to be distinguished from the Bilateral results with a p-value of 0.12. However, combined with the MeanGaussian filter it is distinct from all the other distributions at the 95% and 99% confidence level respectively. The tendency for BM favours the MeanGaussian and Gaussian with a 99% confidence on all but the MeanScharr and MeanLoG filters. The results on SGBM are affected by the timeout method and the consistency in the distributions cannot easily be relied upon for statistical analysis given the number of points.

4.4.2 Analysis

A pareto front approach as seen in Figure 4.23 was used to analyse the performance trade-off in regards to the potential gain in hypervolume when using the additional prefiltering. The plot does not include the filter labels due to overlapping boxes. Non-dominated solutions in this plot are the Scharr, Gaussian and MeanGaussian filters, from left to right respectively. Though the Scharr filter is dominated by the unfiltered result. These three filters also represent the pareto optimal filters for the BM algorithm,

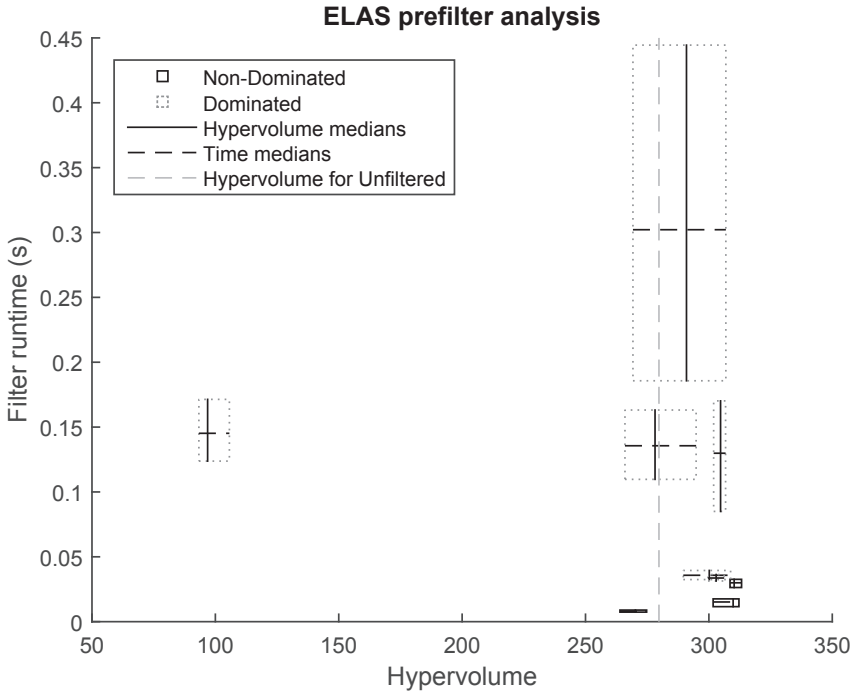


Figure 4.23: An example of the analysis technique used to look at the performance of each prefilter. The boxes represents the 75-25% population of each prefilter as in a boxplot, but the concept is extended to apply along both the hypervolume and time axis. Additionally the pareto front has been generated among the boxes, with the non-dominated filter types having solid borders, whereas the dominated filters have dotted borders. The unfiltered result is included for comparison, but is not heeded in regards to the marked pareto front.

whereas the SGBM algorithm was represented by the Gaussian filter alone. The latter result is likely to have been heavily influenced by the timeout mechanic, as the slightly more computationally heavy MeanGaussian filter will have tipped the timeout threshold when similar SGBM with Gaussian parameters would not. In turn this slowed down the search effort in minimizing the objectives and caused the large variation seen in all SGBM prefilter hypervolume results.

Given the observed potential problems with the timeout mechanic as observed in this and earlier experiments, a more thorough look at the method is in order to ascertain the effect of and need for this mechanic. This will be handled in Section 4.5.

4.5 The Timeout Threshold

Previous experiments had a strict timeout scheme in which slow solutions were stopped prematurely to avoid such solutions from propagating. While this approach has been useful in reducing the total time per run, such a method also reduces genetic diversity and hence may prevent potentially good gene combinations from affecting the future search effort. This experiment aims to look at the necessity of the timeout features as described in Section 3.2.5, by looking at how these implementations affect both the time taken as well as the final quality of the results. The ELAS algorithm will be used due to its wide parameter dependent runtime, as observed in Figure 3.5.

Earlier experiments, among them Section 4.2.1, highlighted the potential for altering the evolutionary objectives previously used. This lead to a change in this experiment to use Bad and Invalid as the objectives to get a more direct mapping to the desired objective space and its resulting hypervolume. The population size and generation count was kept as in the previous experiments to allow for a more direct comparison, leading to the experiment setup in Table 4.17.

Objectives	**Timeout Method**
	None (∞)
Bad, Invalid (and Runtime)	Hard 1s/3s CPU Time w/Regeneration
	Soft 1s/3s CPU Time
Population	**Generations**
100	200
Dataset	**Algorithm**
Training 20	ELAS

Table 4.17

Up till now the timeout threshold has been set some distance above the desired algorithm runtime, this experiment will also test the effect of setting the threshold to the near real-time threshold of 1 second to see the effect

on the result and whether the Soft timeout method is able to counteract the loss of diversity observed (see Section 4.4) with the hard method while keeping the time of each run passable.

With this in mind eight experiments were configured to test the various combinations of the timeout methods. The first five used the Bad and Invalid objectives with three of these testing the Hard timeout method on different thresholds (1s/3s/none), and the remaining two the Soft timeout method (1s/3s).

With the possiblity of adding additional objectives it is also interesting to see whether having a runtime objective enables the search to compensate for the lack of a hard timeout while preserving the diversity required for a good result. To test this, three experiments were included with a runtime objective in addition to the Bad and Invalid objectives.

The setup for this experiment included 4 workers, each calculating the group of 8 methods in cyclical order 3 times over giving a total of 12 runs per variant. To further reduce the variance in the output times all runs were done on the same large server with the aim of reducing the problem seen in Section 4.2.3.

4.5.1 Results

A boxplot of the Bad-Invalid hypervolume generated for these experiments can be found in Figure 4.24. The groups with unspecified types are with the Hard timeout method. To get a better idea of the near real-time individuals found the boxplot also includes the sub-1s individuals of each group which will be referred to as the constrained or near real-time population in this section. The two-tailed Wilcoxon ranksum comparison of the near-real time distributions of each method can be found in full in Table C.1. For each method the corresponding time per evolutionary run can be found in Figure 4.25.

For the unconstrained population the infinite evaluation time method shows significantly better hypervolume than all other methods with P<=0.004 for all cases. However, its runtime is 2-6 times slower than the other methods making for a very time consuming search, and individuals poorly suited for comparison with current official results. Constraining its population to the near real-time individuals only causes the method to fall significantly behind all the other two-objective methods in both hypervolume and time used. With increased evolutionary pressure towards runtime, adding a timeout method gives a significant boost to the near real-time population with all the two-objective methods performing well.

Pruning the population of the 3s Soft method shows increased variance over the other two-objective timeout methods, but the distribution can not be fully distinguished from the 3s Hard timeout with a P-value of 0.1. However, the largest total hypervolume among the fast solutions was created with this 3s Soft method.

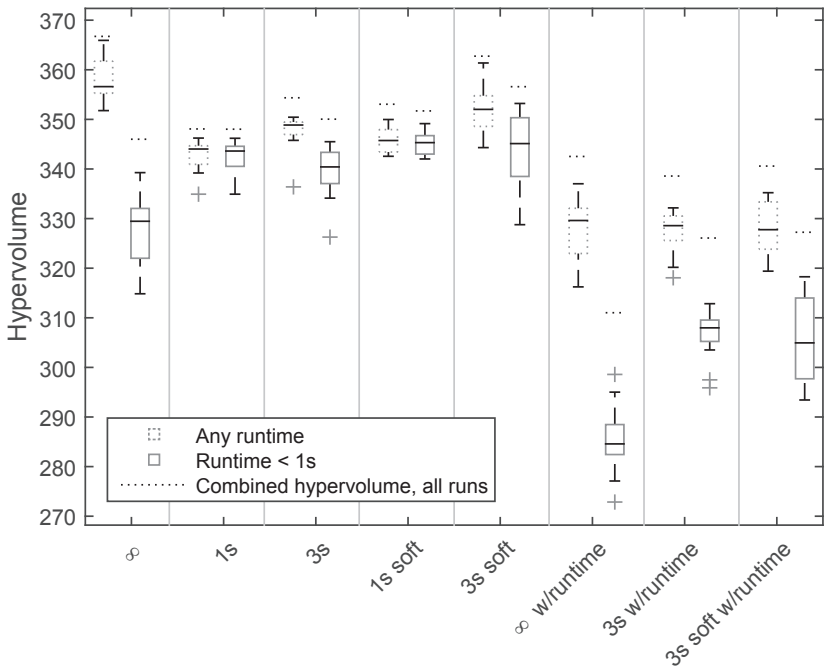


Figure 4.24: A comparison of the Bad-Invalid hypervolume generated for each of the selected timeout methods and objective types. The boxplot shows both the unconstrained raw output of the search and the populations pruned to sub-1s individuals only. Also shown is the total hypervolume if all runs in that bracket are combined and a total pareto front is generated.

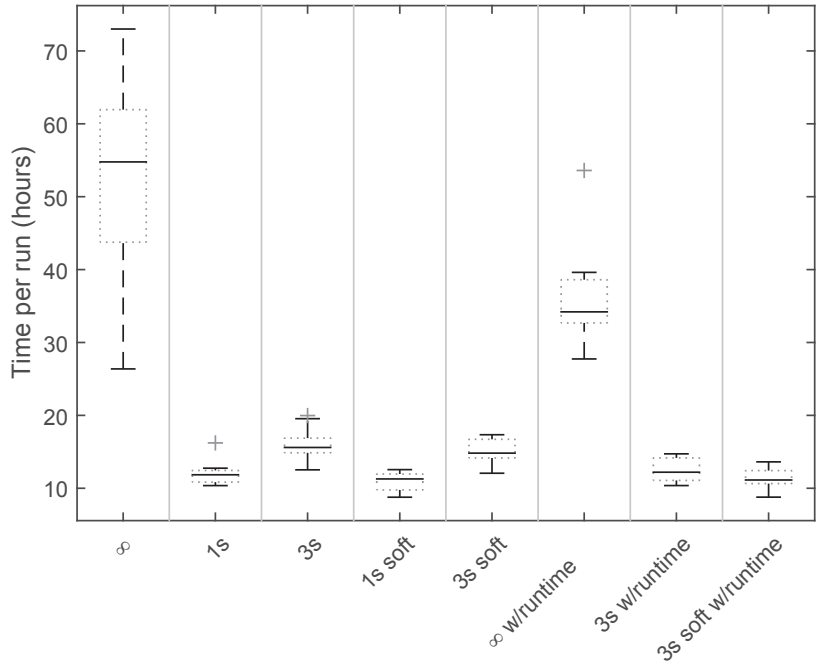


Figure 4.25: Boxplot of the time used per run for each timeout method. The hard timeout methods also include the time spent on evaluating and regenerating new individuals when the previous one failed the timeout threshold.

4.5.2 Analysis

The runtime analysis of Figure 4.25 gives credence to the use of an evaluation time threshold or a runtime objective, as without either the evolutionary approach will have no selection pressure towards selecting faster individuals causing the time per run to spiral upwards. However, given the loss in quality when a runtime objective is included it is desirable to select amongst the timeout methods, unless a large enough population can be included to compensate for the increase in dimensionality gained by the additional objective.

An ideal timeout may not have been found as it requires more runs, but given the good performance of the Soft timeout with the 3 second threshold it will be the method of choice going forward. The longer tail of this method in comparison to the other methods is assumed to give a better defined pareto front when multiple runs are combined.

4.6 Improving on the Results

This section aims to combine the lessons in the previous experiments with the aim of even better results worthy of comparing with the KITTI benchmark. The setup this time can be found in Table 4.18. Changing the objective to Bad and Invalid should allow a more detailed front to be developed, though at the cost of less immediate control of the run time. Though the latter should be encouraged through the use of the soft timeout mechanic.

Objectives	**Timeout Method**
Bad and Invalid	Soft 3s CPU Time
Population	**Generations**
500	100
Dataset	**Verification**
Training 40	Offline

Table 4.18: The setup for the final pareto front experiment.

Based on the early convergence behaviour seen previously the number of generations was cut in half. Additionally the dataset was doubled in size and the population size was increased five-fold, with the goal of better defining the front. These changes meant a total increase in evaluation count by a factor of five. Additionally, the extended parameter set designed in Section 4.2.2 will be used in this experiment to give the genetic algorithm greater freedom and potentially enhancing the results.

To reduce runtime variation and the problem with timeout thresholds this experiment will be limited to two servers each equipped with two Intel Xeon L5640[10] (24 threads per server). These servers were on the

[10] ark.intel.com/products/47926/Intel-Xeon-Processor-L5640-12M-Cache-2_26-GHz-5_86-GTs-Intel-QPI

high end on the number of threads available allowing a quicker overall turnaround. However, after the fact, they were found to be not as powerful on a per individual basis as evident in the results of Section 4.2.3 and 4.2.5. These servers have also been responsible for the test set verification, hence it is expected that the large variations seen in those experiments will be prevented. The reduced number of workers will however severely reduce the number of possible runs within the time allotted for this experiment.

All algorithms were given the MeanGaussian prefilter which previously provided the good results seen in Section 4.4. This was based on the assumption that the machines used with the Soft timeout would give good results even for the SGBM algorithm.

4.6.1 Results

Table 4.19 summarizes the number of runs and time taken for comparison with the MeanGaussian results in Table 4.16. With a five-fold increase in the number of evaluations a similar slowdown can be expected when comparing the MeanGaussian results with the new runs. This is true for the ELAS algorithm with variance and the expected extra NSGA-II overhead able to account for the small additional increase. The other two algorithms show less of a slowdown indicating either a runtime bias in the workers used on the original runs, or a better ability to scale given the number of available threads.

	BM	ELAS	SGBM
Runs	4	4	2
Median hours per run	9.05	86.97	101.47
Relative slowdown	3.39x	5.42x	2.52x

Table 4.19: The number of runs and their median time used per run. Also included is the median slowdown in comparison to the previous results of Table 4.16.

Four runs were initially scheduled for each algorithm with this setup tested on the same machine architectures. However, the last two SGBM runs were not completed as the first two had shown a total lack of convergence, indicating a problem with the setup for this method.

The pareto front plot of Figure 4.26 shows the combined pareto fronts of the three algorithms as computed based on all their runs. The corresponding dotted lines illustrates the total front in the MeanGaussian prefilter experiment of Section 4.4. A marked improvement can be seen on the BM and ELAS fronts.

Figure 4.27 contains boxplots of the hypervolume per run. Again for comparison to the previous result, the median MeanGaussian hypervolume is included by a dotted line. Comparing the distributions of MeanGaussian with the new runs gives a significant increase for the BM and ELAS algorithms. Both are significant at the 99% confidence level with a P-value of 0.004 and 0.008 respectively. The SGBM results can not be

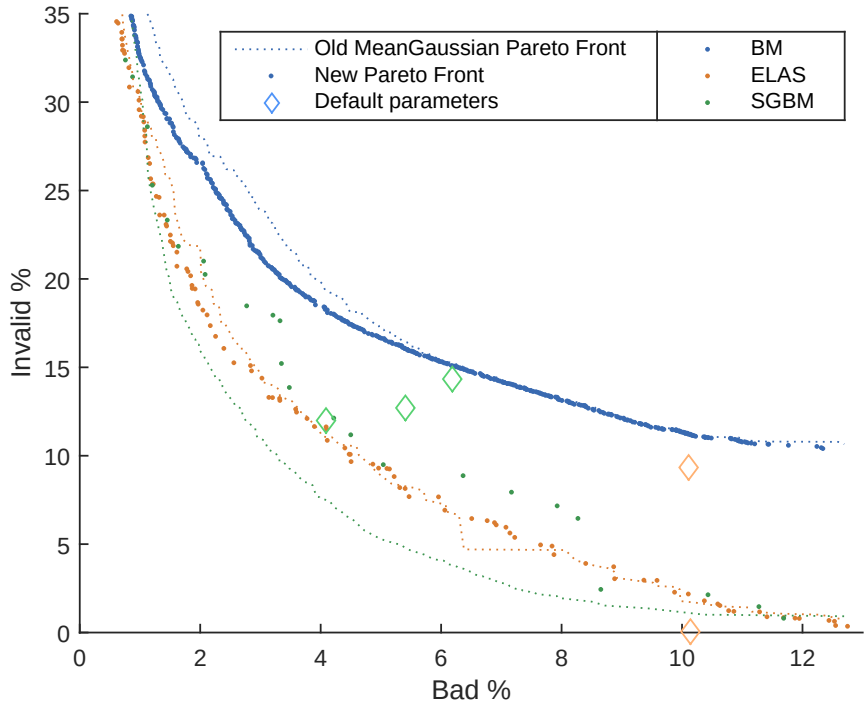


Figure 4.26: The combined pareto fronts in the Bad-Invalid space on the test set. Result of the equivalent filter of Section 4.4 included for comparison.

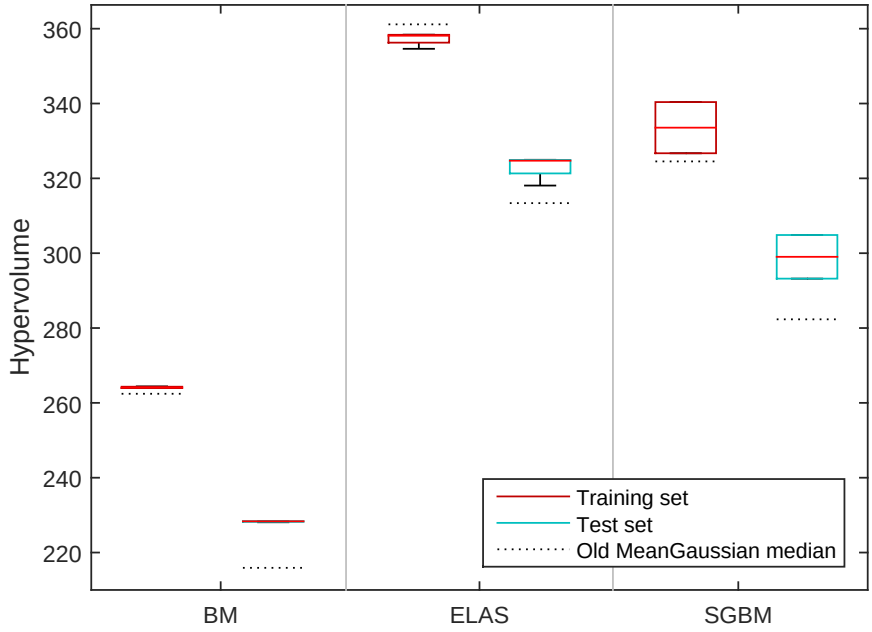


Figure 4.27: Boxplot of the Bad-Invalid hypervolume, as generated from each run of Section 4.6

distinguished from the previous distribution with a P-value of 0.64.

Combined Hypervolume	BM	ELAS	SGBM
This Experiment	229.1	328.1	309.6
MeanGaussian Experiment	219.6	325.3	352.9
Baseline	203.9	293.0	349.5
Median Hypervolume	BM	ELAS	SGBM
This Experiment	228.3	324.7	299.03
MeanGaussian Experiment	215.9	313.4	282.3
Baseline	202.1	283.1	345.8

Table 4.20: A Comparison of the Bad-Invalid Hypervolume with earlier experiments. The MeanGaussian data is from Section 4.4, the Baseline is from Section 4.2.1.

	Default	Target Invalid		As Their Best Default	
	(Bad, Invalid)	15%	5%	Similar Bad	Similar Invalid
BM	(6.92, 45.60)	(**6.28**, 15.00)	-	(6.88, **14.31**)	(**0.39**, 45.45)
ELAS	(10.11, 9.30)	(**2.85**, 14.8)	(**7.65**, 4.96)	(10.10, **2.18**)	(**4.96**, 9.30)

Table 4.21: An updated version of the potential quality-density trade-offs previously seen in Table 4.12. Target Invalid refers to keeping the density constant at the given value then looking for what quality that entails along the pareto front. As Best Default shows the density and quality attainable if respectively Bad or Invalid are locked to the current algorithm default result.

4.6.2 Analysis

The changes done to the evolutionary setup in this experiment improved both the consistency of the hypervolume per run and the total definition of the combined pareto front. Given that both the parameter ranges and the training set size was altered, it is not possible to easily answer the effect of each. However, as evident by the ELAS MeanGaussian outlier value visible in Figure 4.26 there is still room for improvement. There is naturally a random element at work, but a more focused search within the region of interest as per the reference point strategy of NSGA-III may have been beneficial to search progress.

The new pareto optimal trade-offs presented in Table 4.21 shows a marked improvement in comparison with the Baseline created in Section 4.2.1. Selecting the individuals and submitting the results will be handled in Section 4.7.

To better explain the SGBM results, its complete search space was plotted in Figure 4.30. Compared to the objective space plots of earlier experiments this newer population shows a complete lack of convergence towards the pareto front. Looking at the output of the internal NSGA-II state showed all individuals marked with a constraint violation of

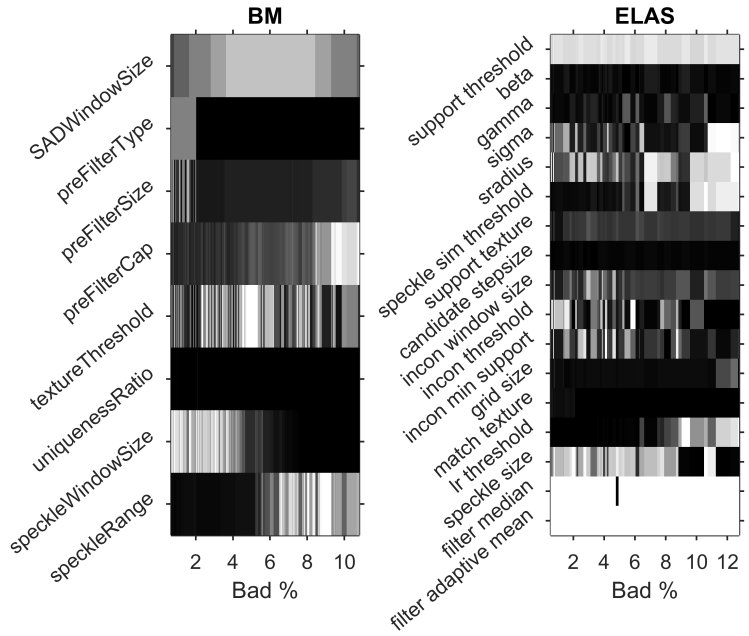


Figure 4.28: Histograms showing the parameters of the stereo algorithms for the final experiment. SGBM not included as it did not converge properly. The colour encoding matches the rules of Figure 4.9, but note the change in max parameter ranges on certain parameters.

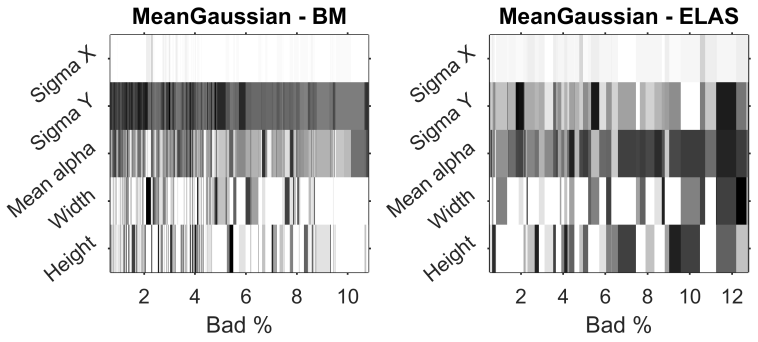


Figure 4.29: Histograms showing the parameters controlling the MeanGaussian prefilter along each part of the pareto front. The allowed ranges were sigma: 0-1, mean: 0-255, width/height: 3-19.

varying size caused by the Soft timeout mechanic giving a penalty to slow individuals. This would normally not be a problem, but as the machines used in this experiment were unable to compute *any* SGBM instance within the soft timeout, this meant the entire population was in violation of the threshold and marked accordingly. Based on the NSGA-II implementation this in turn meant that the dominance metric was no longer used within the tournament selection step, as with two constraint violating individuals the implementation will always select the least violating. With the entire population involved this caused no evolutionary pressure towards the selected objectives and the search progress faltered. In fact the only pressure exhibited was towards the runtime giving in essence a single-objective search, which worked well enough in its regard with 94.6% of the evaluated population favouring the faster 5-way programming. A better approach for the SGBM algorithm would have been to disable the timeout and potentially locking the *mode* parameter to the 5-way dynamic programming if further speed was required. This would have found the pareto optimal solutions without machine-dependent runtime concerns.

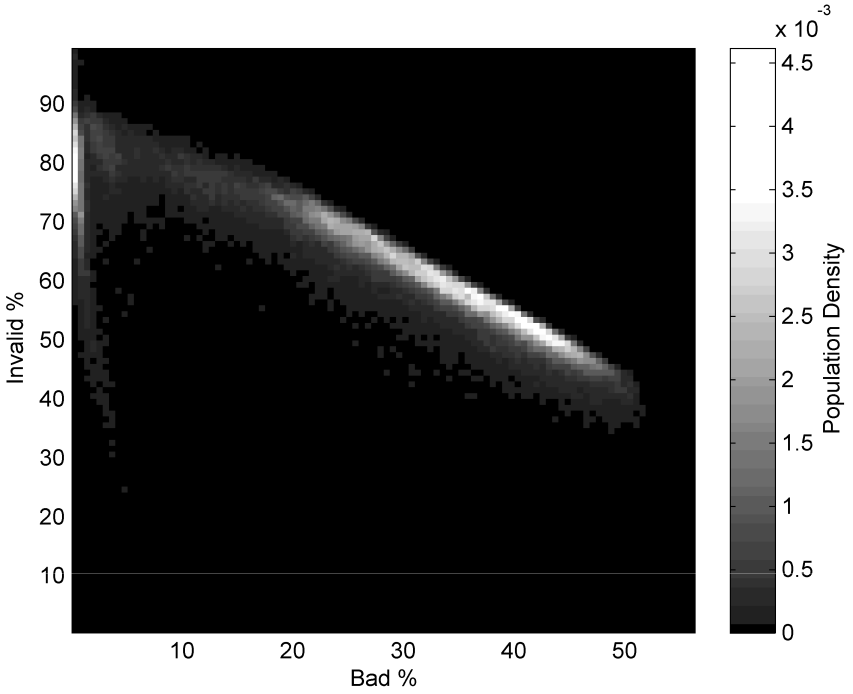


Figure 4.30: Analysis of the SGBM objective space resulting from the experiment in Section 4.6.

4.7 Submitting Results to the KITTI benchmark

The KITTI benchmark provides an option to submit ones own results and have them evaluated on their internal test set. This requires running the desired algorithm on the 195 KITTI test set images then posting them to the site. Accomplishing this was done via reusing the ability of the framework

to load individuals from parameter files, originally used for the offline pareto front method. This required only a small modification wherein the disparity evaluation was skipped and the calculated disparity images were saved to disk using the same 16-bit PNG image format as the KITTI ground truth images for support with the benchmark.

Submitting results has a limit on once every 72 hours, as such the number of tests was kept to just one per algorithm.

For BM the current best published result is very sparse with just 55.84% of the image computed. While a similarly dense solution could be submitted, the pareto front on earlier experiments suggests a denser result could be provided. Given the shape of the pareto curve an Invalid rating of 15 percent may be too strict, so a target of 20 percent will be used from the results of Figure 4.26 as that front dominates the other experiments along its entirety. The BM pareto front is heavily populated so looking at the 20 percent region presents many similar candidates. An individual was selected by introducing the RMS error as a third axis then selecting a low RMS individual within the 20 percent Invalid region. As the pareto front is generated based on Bad and Invalid this also meant a low Bad for this individual.

To provide a fair ELAS individual the selection will be concentrated in the region of 95% density (5% Invalid) thereby matching the currently published solution. The prefilter MeanGaussian experiment and the latest experiment of Section 4.6 both contribute towards the total pareto front hence will be combined before final individual selection. The MeanGaussian outlier clearly visible in front of the pareto front plotted in Figure 4.26 makes for an interesting trade-off and will be the choice going forward with this algorithm. Its expected performance matches that of the currently posted result in density, but should be better in quality if the results transfer well.

SGBM did not produce any results in the final experiment, but several good individuals were found in the Prefiltering experiment of Section 4.4. As such the chosen individual will be from that population. The currently published result has a density of 86.5% hence a matching individual was found. In addition a higher density candidate was also chosen in the 5-7% Invalid region using a similar technique to the one used for the BM selection.

4.7.1 Results

By running the individuals on the KITTI test set the density of the result can be ascertained without having to submit the data. This showed that a large difference exists in the density for both the BM and SGBM individuals when transferred to the KITTI test set. The SGBM individual purported to be of similar density to the official results turned out to produce 26.9 Invalid. As this is significantly less than the current official result of 13.5, the alternate individual was picked for submission. The individuals picked are listed with their Framework statistics in Table 4.22, with their actual parameter values listed in Appendix C.2.

The output of the selected individuals were submitted to the benchmark giving the summarized results of Table 4.23. A summary of the current official algorithm results is also included. The Density posted on the benchmark has been converted to the Invalid objective (100%-Density) in this table. The full benchmark outputs including the current official results are included in Appendix C.2. The runtime data for the results from this thesis are in CPU time from the usual verification server.

Framework Results

	BM	ELAS	SGBM
Bad	3.26	6.38	4.38
Invalid	20.35	4.71	6.72
Runtime (s)	0.21	1.43	4.37
AvgErr	0.95	1.20	1.14

Table 4.22: The individuals selected for testing on the official KITTI testset. Data is the current Framework test set output.

	Thesis Results			**Current Offical**		
	BM	ELAS	SGBM	BM	ELAS	SGBM
Bad	9.65	6.72	6.09	25.38	8.24	7.64
Invalid	32.66	7.58	11.23	44.16	5.45	13.5
Runtime (s)	0.11	0.88	0.82	0.1	0.3	1.1
AvgErr	1.7	1.2	1.4	7.6	1.4	1.8

Table 4.23: A summary of selected individuals and their framework output and their KITTI benchmark results. For comparison the current officially posted results for the active algorithms are also included. Grey cells indicate the better result.

The benchmark presents the full results on the first 20 test set images and gives a summary of the total 195 image results. The output for the Bad objective on the first 20 images with both the individuals from this thesis and the official current results are presented in Figure 4.31. The median improvement for these images on the Bad metric ($\frac{Bad_{Old}}{Bad_{New}}$) was 2.82, 1.22 and 1.46 for the BM, ELAS and SGBM algorithms respectively.

4.7.2 Analysis

As per Table 4.23, the submitted individuals provided a significant improvement over the current defaults. The image-wise plot of Figure 4.31 showed a massive improvement for the BM algorithm in particular with some cases improving by up to 4.7x over the current results. SGBM and ELAS showed more localized betterment of results with up to 3.4x improvement in certain cases and the occasional small drop in quality. As seen in the plot certain images are more difficult to compute. This could be because of a lack of training data covering such a scenario, but for at least some of the presented cases they can be explained by common

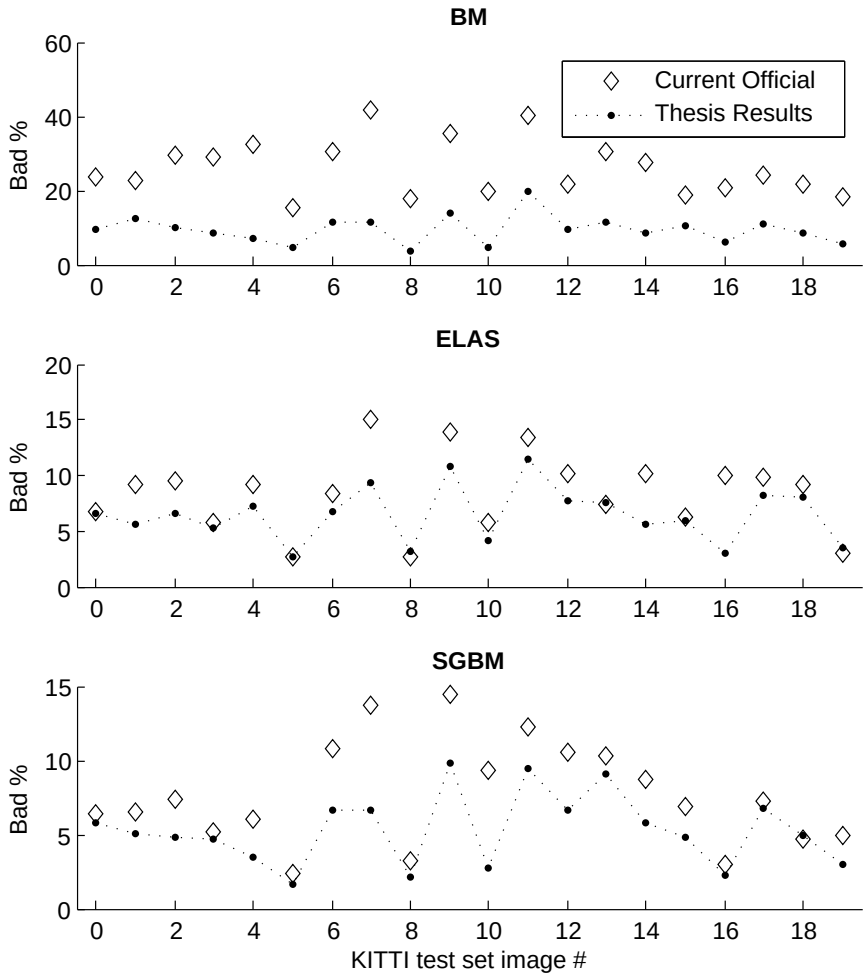


Figure 4.31: The results on the first 20 images of the KITTI test set for each of the three algorithms tested. The current official results are represented by a diamond, while the marked dotted line is the results of the individuals submitted for this thesis.

stereo matching problems. As an example images 9 and 11 are consistently difficult for the algorithms to compute. A large part of both image pairs contain reflective surfaces in the form of parked cars and there is also a lack of texture in certain areas causing matching to be difficult.

When saving the output images the unknown disparity value marking uncalculated pixels is transformed into the value 0 in the image. Any existing disparities of exactly 0 will also be marked by the same value. Hence it is possible that a small loss in density is caused by this implementation specific issue.

A loss in runtime is observed when going from the current official results to the thesis results for the ELAS algorithm. While the thesis evaluations have been done on a slightly slower computer in pure GHz terms it is likely that the more fine-grained *grid size* of the chosen parameters represent a heavier load than the default. The chosen SGBM individual uses the same 8-way dynamic programming, yet even with the additional prefilter it outperforms the official results. This discrepancy is likely caused by the computational optimisations added in newer OpenCV versions.

In Section 4.2.5 an assumption was made based on the runtime within the framework as compared to the single image pair at a time runtime seen by algorithms in normal use. For the selected individuals this assumption held as can be seen by comparing Table 4.22 with Table 4.23.

An interesting observation with the SGBM individual picked is that it disables its left-right disparity check. However looking at the output images from the algorithm shows a similar apparent effect just via a different internal mechanism. Adjusting the parameter set to include the default LR-check of 2 actually increased the number of marked pixels while decreasing their accuracy as evaluated on the (framework) test set. This may seem counter-intuitive given that the LR-check can only remove pixels, not add, but in this case it is explained by the smaller speckle areas being created during matching causing the evolved speckle parameters to trigger less often. As such these areas are not removed and the overall quality is reduced slightly.

This prompted an investigation into the BM algorithm as well, and a similar ability to reject unreasonable matches is seen there. Looking at the parameter values gives the indication that the speckle removal post-processing step is used to remove these regions in much the same way the LR-check would have done. This would explain the improved results when these parameters were increased to such a large extent when the extended parameter set was implemented (Appendix A). It also shows how the BM algorithm could produce good results despite being forced to disable the LR-check due to the observed instability issues.

Chapter 5

Discussion

In this chapter the thesis is concluded with a general discussion of the results attained, the thesis conclusion, and possible future work in relation to the subject.

5.1 General Discussion

This section will discuss the thesis results generated, and some of the specifics on the results of the framework modules and their performance.

5.1.1 Evolutionary Multi-Objective Optimisation

The results in Section 4.2 showed the potential for using the implemented framework for parameter optimisation, with tested cases of up to 17 control parameters optimised successfully. The found parameter distributions along the pareto fronts, as discussed in Section 4.2.2, indicated the inherent problem with a manual optimisation procedure based on the variety in the evolved parameters. This main experiment also displayed a promising method of algorithm comparison using the multi-objective method. This gave not only the resulting quality metric, but also the actual density and the potential trade-offs amongst the two. With the algorithm results interpolation turned off, this allowed direct comparison of the potential pareto fronts of each algorithm, which may be of particular interest if the default author-provided trade-off is not supportive to the desired use case. Within this experiment, a comparison was also made with the scalarisation approach to optimisation, wherein the very localised results of the latter was presented. The main approach was later expanded on in Section 4.4 with evolution of a prefiltering module and successful optimisation of up to 23 control parameters.

It is evident, based on the results on the last experiment of Section 4.7, that the framework was able to create robust parameter sets for the images of the KITTI benchmark. The parameters were able to improve upon the currently held algorithm defaults for all the three tested stereo algorithms. The framework was not however without its faults, which will be discussed in separate sections.

5.1.2 The Framework as a Design Tool

Section 4.4 introduced the possibility of changing the algorithms by altering a component part in the form of an extra prefiltering step. By testing and plotting a multitude of options, an improving strategy was found and included in further experiments. This presents an interesting potential workflow for an algorithm designer, allowing the full prospective quality-density trade-offs of each change to be considered before a final design decision is made.

The analysis of the final submitted results (Section 4.7) provided an interesting insight into parameter usage most likely unintended by the algorithm authors. Other parameters were evolved into taking the job of the left-right disparity check, which the evolutionary approach found beneficial, as it was able to improve on the results it was otherwise able to attain.

5.1.3 The Choice of Objectives

The choice of objectives in the main experiments of Section 4.2, 4.4 and 4.6 were able to control the evolution into creating good results able to beat the current default parameters.

However, the initial objective experiment (Section 4.1.3) discounted the possibility of trying RMS & Invalid, or avgErr & Invalid as possible choices, as these objectives were poorly represented by the validation method used at the time. This led to the choice of a potentially too lax Bad threshold, with little evolutionary pressure towards gaining results more precise than 3 disparity levels from the ground truth. While the ELAS and SGBM individuals finally submitted were evolved with an additional RMS objective, the BM individual was not, giving the possibility of its larger 3-2px level gap, evident in Table C.3, being caused by this large threshold. A stricter threshold of 1 or 2 levels, or a change in objectives, is likely to have been able to transfer better when the algorithm was later rated on the KITTI benchmark.

5.1.4 The Time Aspect

One of the main problems encountered within this thesis was in relation to the time used by the algorithms themselves. In Section 4.1.1 a parallel approach was tested and found favourable in comparison to the very slow serial method. While allowing the number of runs to increase, this change also lead to a mismatch between the runtimes recorded within the framework and the time these parameters would use outside of it. This was caused by the algorithms slowing down when other independent algorithm instances were computing in parallel. As the different algorithms were differently affected, also in regard to differing worker machines, this made metrics depending on the runtime unreliable. In turn the timeout methods initially tuned to one machine architecture became overly active

on others, causing poor search progress on some workers (Section 4.4) or even indirectly changing the active optimisation objective (Section 4.6).

5.2 Conclusion

A framework for stereo algorithm parameter optimisation was developed for this thesis based on the multi-objective evolutionary algorithm NSGA-II. Over a series of experiments testing different aspects of the implementation, the specifics of the framework were refined. Tests were performed wherein up to 23 separate stereo control parameters were simultaneously optimised. The multi-objective approach presented a multitude of evolved potential algorithm parameter sets in its generated pareto fronts. This allowed a designer or user to pick a desired trade-off with increased knowledge of the possible objective space, setting it apart from the more traditional manual optimisation, or single-objective scalarisation approach.

Via the generated pareto fronts of each algorithm, it became simple to select a desired trade-off in the quality-density space. This allowed three parameter sets, each able to match the current official results on density, to be selected for submission to the KITTI test set using its official evaluation tools. A single optimised parameter set was turned in for each algorithm, with promising and more robust results than the currently posted official algorithm parameters. This showed the potential of the multi-objective parameter optimisation approach, and its implementation in the framework as developed for this thesis. These individuals also showed a novel use of parameters, wherein the evolutionary approach was able to discover alternate uses for some of the parameters by using them in unintended ways.

The runtimes of the stereo algorithms provided a mayor challenge due to the cost of evaluating the training data used to assign individual fitness. Several methods were implemented to reduce the total time per evolutionary run, and additionally improve the relevance of the resulting individuals in regards to their final stereo image throughput. More research is however required to increase the fairness and relevance of the recorded evaluation runtime when multiple worker machines are in use.

5.3 Future Work

A potential partial solution to the runtime variance observed would be a benchmarking step of each worker. This would in part be able to compensate for differing worker performance as affected by e.g. pipelining, CPU speed, memory architectures and system load. The work of Tippets et al. [72] mentions such a runtime normalization factor, but also cautions the difficulty of achieving fairness in some cases. It is however likely that within-algorithm fairness could be achieved via this technique, and that the timeout methods employed would be able to properly guide the search towards their intended runtime threshold.

Due to the problems with the runtime metric a more thorough comparison of algorithms in runtime space was not practically possible. For algorithms featuring a very flexible parameter dependent runtime it may be interesting to look at the possible hypervolume for each time target. By creating such a hypervolume-to-runtime plot for each algorithm the faster algorithms should become better represented in their region of speciality than current benchmarks may indicate.

The time per evaluation in combination with the number of evaluations meant that some of these experiments have been particularily time consuming (see e.g. Table 4.16). A promising possibility is using multi-objective optimisation algorithms specifically developed for problems with expensive evaluations. The Bin_MSOPS [41] and ParEGO [39] algorithms may provide such a potential, comparing favourably with NSGA-II. However, as the scale of the objectives need to be known in advance, the runtime as an objective may need to be avoided or have a threshold applied.

The regenerator module used in this thesis randomly recreated failed individuals generated by the timeout methods and uniqueness checker. A more targeted approach wherein new individuals are attempted created in currently unexplored regions may prove a better method. This may take the form of a simple crossover between individuals of interest, or a more parameter tuned approach as in the search space optimisation of the above budget multi-objective optimisers.

The Uniqueness checker module was able to significantly save evaluation time, but as apparent in Table 4.13, it was of less use when real-valued parameters were introduced. While this could simply be because of the large search space of the ELAS algorithm, a similar tendency was observed when prefilters with real-valued genomes were added to the later experiment. As such, a better similarity measure should be developed and employed. This measure must take into account the possibility of certain parameters effectively disabling others, as is the case of some prefiltering parameters, thereby leading to different genomes causing the same practical evaluation. This is likely best done in unison with the algorithms and the prefilters by letting their translation interfaces sort out whether a gene contributes and whether the genomes are practically identical.

Based on the comparison of the runtimes of BM and ELAS when evaluated in parallel and in regards to their true serial performance (Comparison in Section 4.7), they are well suited for an approach wherein multiple configurations are evaluated at once. Given that individuals along the front can specialize in different stereo subproblems, as observed in Section 4.2.1, this may allow a stereo fusion approach wherein multiple stereo instances all contribute to parts of the results in a weighted sum. This may be handled as in a knapsack-problem with a fixed runtime roof and individuals picked across the current pareto fronts, with their results combined based on their training set results.

The brand new 2015 KITTI stereo benchmark [52], includes the related fields of optical and scene flow into the evaluation. As these too can be evaluated in a similar fashion to the current implementation this presents

the possible expansion of the framework to account for these changes and their new algorithms. However, at the time of publication, only one of the currently listed algorithms handling all three problems have runtimes within the ranges of the algorithms tested by the framework in this thesis, thereby limiting the testing potential.

The Prefiltering experiment of Section 4.4 showed the potential for evolutionary optimising and choice of pre-processing module for a stereo algorithm. If this concept were to be extended to also include the internals of the stereo algorithms in a stereo building block approach. Various matching cost computations, cost aggregation, disparity optimisation and post-processing methods, could all be controlled by an evolutionary approach. This may seek inspiration from the modular stereo framework used in Scharsteins taxonomy [66], with added evolutionary control. Combined with a method-labeled pareto front output, this may improve understanding of the methods and create better algorithms in the future. However a more flexible framework will have to be created to account for the variously sized genomes representing the variety of internal modules.

In this thesis the actual evolutionary control parameters of the crossover and mutation operators have not been adjusted to better suit the problem. It is likely that such a problem specific modification would allow shorter generational runs and further reduced variance between runs.

Bibliography

[1] Adnan Ansar, Andres Castano and Larry Matthies. 'Enhanced real-time stereo using bilateral filtering'. In: *3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on*. IEEE. 2004, pp. 455–462.

[2] Herbert Bay, Tinne Tuytelaars and Luc Van Gool. 'Surf: Speeded up robust features'. In: *Computer vision–ECCV 2006*. Springer, 2006, pp. 404–417.

[3] Stan Birchfield and Carlo Tomasi. 'Depth discontinuities by pixel-to-pixel stereo'. In: *International Journal of Computer Vision* 35.3 (1999), pp. 269–293.

[4] Tobias Blickle and Lothar Thiele. *A comparison of selection schemes used in genetic algorithms*. 1995.

[5] Yuri Boykov, Olga Veksler and Ramin Zabih. 'Fast approximate energy minimization via graph cuts'. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 23.11 (2001), pp. 1222–1239.

[6] Gary Bradski. 'The opencv library'. In: *Doctor Dobbs Journal* 25.11 (2000), pp. 120–126.

[7] Lawrence Davis. 'Job shop scheduling with genetic algorithms'. In: *Proceedings of an international conference on genetic algorithms and their applications*. Vol. 140. Carnegie-Mellon University Pittsburgh, PA. 1985.

[8] Kalyanmoy Deb and Hans-georg Beyer. 'Self-adaptive genetic algorithms with simulated binary crossover'. In: *Complex Systems*. Citeseer. 1995.

[9] Kalyanmoy Deb and Mayank Goyal. 'A combined genetic adaptive search (GeneAS) for engineering design'. In: *Computer Science and Informatics* 26 (1996), pp. 30–45.

[10] Kalyanmoy Deb and Himanshu Jain. 'An evolutionary many-objective optimization algorithm using reference-point-based non-dominated sorting approach, part i: Solving problems with box constraints'. In: *Evolutionary Computation, IEEE Transactions on* 18.4 (2014), pp. 577–601.

[11] Kalyanmoy Deb et al. 'A fast and elitist multiobjective genetic algorithm: NSGA-II'. In: *Evolutionary Computation, IEEE Transactions on* 6.2 (2002), pp. 182–197.

[12] Agoston E Eiben, P-E Raue and Zs Ruttkay. 'Genetic algorithms with multi-parent recombination'. In: *Parallel Problem Solving from Nature—PPSN III*. Springer, 1994, pp. 78–87.

[13] Pedro F Felzenszwalb and Daniel P Huttenlocher. 'Efficient belief propagation for early vision'. In: *International journal of computer vision* 70.1 (2006), pp. 41–54.

[14] Mark Fleischer. 'The measure of Pareto optima applications to multi-objective metaheuristics'. In: *Evolutionary multi-criterion optimization*. Springer. 2003, pp. 519–533.

[15] Carlos M Fonseca, Peter J Fleming et al. 'Genetic Algorithms for Multiobjective Optimization: Formulation Discussion and Generalization.' In: *ICGA*. Vol. 93. 1993, pp. 416–423.

[16] Michael Frigge, David C Hoaglin and Boris Iglewicz. 'Some implementations of the boxplot'. In: *The American Statistician* 43.1 (1989), pp. 50–54.

[17] Pascal Fua. 'A parallel stereo algorithm that produces dense depth maps and preserves image features'. In: *Machine vision and applications* 6.1 (1993), pp. 35–49.

[18] Andreas Geiger, Philip Lenz and Raquel Urtasun. 'Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite'. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[19] Andreas Geiger, Martin Roser and Raquel Urtasun. 'Efficient large-scale stereo matching'. In: *Computer Vision–ACCV 2010*. Springer, 2011, pp. 25–38.

[20] Andreas Geiger et al. 'Vision meets Robotics: The KITTI Dataset'. In: *International Journal of Robotics Research (IJRR)* (2013).

[21] Steven B Goldberg, Mark W Maimone and Larry Matthies. 'Stereo vision and rover navigation software for planetary exploration'. In: *Aerospace Conference Proceedings, 2002. IEEE*. Vol. 5. IEEE. 2002, pp. 5–2025.

[22] Minglun Gong and Yee-Hong Yang. 'Genetic-based stereo algorithm and disparity map evaluation'. In: *International Journal of Computer Vision* 47.1-3 (2002), pp. 63–77.

[23] Raj Kumar Gupta and Siu-Yeung Cho. 'A correlation-based approach for real-time stereo matching'. In: *Advances in Visual Computing*. Springer, 2010, pp. 129–138.

[24] Marsha Jo Hannah. *Computer matching of areas in stereo images*. Tech. rep. DTIC Document, 1974.

[25] Heiko Hirschmuller. 'Stereo processing by semiglobal matching and mutual information'. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 30.2 (2008), pp. 328–341.

[26] Heiko Hirschmüller, Maximilian Buder and Ines Ernst. 'Memory efficient semi-global matching'. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 3 (2012), pp. 371–376.

[27] Heiko Hirschmuller and Daniel Scharstein. 'Evaluation of cost functions for stereo matching'. In: *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE. 2007, pp. 1–8.

[28] Heiko Hirschmüller and Daniel Scharstein. 'Evaluation of stereo matching costs on images with radiometric differences'. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 31.9 (2009), pp. 1582–1599.

[29] John H Holland. 'Genetic algorithms and the optimal allocation of trials'. In: *SIAM Journal on Computing* 2.2 (1973), pp. 88–105.

[30] John H Holland. 'Outline for a logical theory of adaptive systems'. In: *Journal of the ACM (JACM)* 9.3 (1962), pp. 297–314.

[31] Asmaa Hosni et al. 'Local stereo matching using geodesic support weights'. In: *Image Processing (ICIP), 2009 16th IEEE International Conference on*. IEEE. 2009, pp. 2093–2096.

[32] Asmaa Hosni et al. 'Real-time local stereo matching using guided image filtering'. In: *Multimedia and Expo (ICME), 2011 IEEE International Conference on*. IEEE. 2011, pp. 1–6.

[33] Andrew Howard. 'Real-time stereo visual odometry for autonomous ground vehicles'. In: *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE. 2008, pp. 3946–3952.

[34] Stephen S Intille and Aaron F Bobick. *Disparity-space images and large occlusion stereo*. Springer, 1994.

[35] Wilfried Jakob, Martina Gorges-Schleuter and Christian Blume. 'Application of Genetic Algorithms to Task Planning and Learning.' In: *Parallel Problem Solving from Nature 2, PPSN-II, Brussels, Belgium, September 28-30, 1992*. 1992, pp. 293–302.

[36] Mikkel T Jensen. 'Guiding single-objective optimization using multi-objective methods'. In: *Applications of Evolutionary Computing*. Springer, 2003, pp. 268–279.

[37] Michael L Kaiser et al. 'The STEREO mission: An introduction'. In: *The STEREO Mission*. Springer, 2008, pp. 5–16.

[38] Alexandre Karpenko et al. 'Digital video stabilization and rolling shutter correction using gyroscopes'. In: *CSTR* 1 (2011), p. 2.

[39] Joshua Knowles. 'ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems'. In: *Evolutionary Computation, IEEE Transactions on* 10.1 (2006), pp. 50–66.

[40] Joshua Knowles and David Corne. 'The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation'. In: *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*. Vol. 1. IEEE. 1999.

[41] Joshua Knowles and Evan J Hughes. 'Multiobjective optimization on a budget of 250 evaluations'. In: *Evolutionary Multi-Criterion Optimization*. Springer. 2005, pp. 176–190.

[42] Joshua D Knowles, Richard A Watson and David W Corne. 'Reducing local optima in single-objective problems by multi-objectivization'. In: *Evolutionary multi-criterion optimization*. Springer. 2001, pp. 269–283.

[43] Kurt Konolige. 'Small vision systems: Hardware and implementation'. In: *Robotics Research*. Springer, 1998, pp. 203–212.

[44] Mario Köppen and Kaori Yoshida. 'Substitute distance assignments in NSGA-II for handling many-objective optimization problems'. In: *Evolutionary Multi-Criterion Optimization*. Springer. 2007, pp. 727–741.

[45] Raphael Labayrade, Didier Aubert and J-P Tarel. 'Real time obstacle detection in stereovision on non flat road geometry through" v-disparity" representation'. In: *Intelligent Vehicle Symposium, 2002. IEEE*. Vol. 2. IEEE. 2002, pp. 646–651.

[46] Zhifeng Liu and Reinhard Klette. 'Approximated ground truth for stereo and motion analysis on real-world sequences'. In: *Advances in Image and Video Technology*. Springer, 2009, pp. 874–885.

[47] David G Lowe. 'Object recognition from local scale-invariant features'. In: *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[48] Santosh Kumar Mandal et al. 'A memetic NSGA-II for the bi-objective mixed capacitated general routing problem'. In: *Journal of Heuristics* (2015), pp. 1–32.

[49] David Marr and Tomaso Poggio. 'Cooperative computation of stereo disparity'. In: *Science* 194.4262 (1976), pp. 283–287.

[50] David Marr and A Vision. 'A computational investigation into the human representation and processing of visual information'. In: *WH San Francisco: Freeman and Company* (1982).

[51] Xing Mei et al. 'On building an accurate stereo matching system on graphics hardware'. In: *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE. 2011, pp. 467–474.

[52] Moritz Menze and Andreas Geiger. 'Object Scene Flow for Autonomous Vehicles'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3061–3070.

[53] Matthias Michael et al. 'Real-time stereo vision: Optimizing semi-global matching'. In: *Intelligent Vehicles Symposium (IV), 2013 IEEE*. IEEE. 2013, pp. 1197–1202.

[54] *Middlebury Stereo Evaluation*. URL: http://vision.middlebury.edu/stereo/eval/ (visited on 29/09/2014).

[55] M Davoodi Monfared, A Mohades and J Rezaei. 'Convex hull ranking algorithm for multi-objective evolutionary algorithms'. In: *Scientia Iranica* 18.6 (2011), pp. 1435–1442.

[56] Filip Mroz and Toby P Breckon. 'An empirical comparison of real-time dense stereo approaches for use in the automotive environment'. In: *EURASIP Journal on Image and Video Processing* 2012.1 (2012), pp. 1–19.

[57] Gerhard Neukum and R Jaumann. 'HRSC: The high resolution stereo camera of Mars Express'. In: *Mars Express: The Scientific Payload*. Vol. 1240. 2004, pp. 17–35.

[58] David Ojeda. *NSGA2 C++ implementation*. URL: https://github.com/dojeda/nsga2-cpp (visited on 05/11/2014).

[59] *OpenCV library*. URL: http://opencv.org/ (visited on 02/02/2015).

[60] Gaurav Pandey, James R. McBride and Ryan M. Eustice. 'Ford campus vision and lidar data set'. In: *International Journal of Robotics Research* 30.13 (Nov. 2011), pp. 1543–1552.

[61] T. Peynot, S. Scheding and S. Terho. 'The Marulan Data Sets: Multi-Sensor Perception in Natural Environment with Challenging Conditions'. In: *International Journal of Robotics Research* 29.13 (Nov. 2010), pp. 1602–1607.

[62] Chuck Pheatt. 'Intel® threading building blocks'. In: *Journal of Computing Sciences in Colleges* 23.4 (2008), pp. 298–298.

[63] Cristiano Premebida, Oswaldo Ludwig and Urbano Nunes. 'LIDAR and vision-based pedestrian detection system'. In: *Journal of Field Robotics* 26.9 (2009), pp. 696–711.

[64] Edward Rosten, Gerhard Reitmayr and Tom Drummond. 'Real-time video annotations for augmented reality'. In: *Advances in Visual Computing*. Springer, 2005, pp. 294–302.

[65] J David Schaffer. 'Multiple objective optimization with vector evaluated genetic algorithms.' In: *Proceedings of the 1st International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 1985*. 1985, pp. 93–100.

[66] Daniel Scharstein and Richard Szeliski. 'A taxonomy and evaluation of dense two-frame stereo correspondence algorithms'. In: *International journal of computer vision* 47.1-3 (2002), pp. 7–42.

[67] Daniel Scharstein and Richard Szeliski. 'High-accuracy stereo depth maps using structured light'. In: *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2003, pp. I–195.

[68] Daniel Scharstein et al. 'High-resolution stereo datasets with subpixel-accurate ground truth'. In: *Pattern Recognition*. Springer, 2014, pp. 31–42.

[69] Yu Shuchun et al. 'Preprocessing for stereo vision based on LOG filter'. In: *Strategic Technology (IFOST), 2011 6th International Forum on*. Vol. 2. IEEE. 2011, pp. 1074–1077.

[70] Karl Sims. 'Evolving virtual creatures'. In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM. 1994, pp. 15–22.

[71] Nidamarthi Srinivas and Kalyanmoy Deb. 'Multiobjective optimization using nondominated sorting in genetic algorithms'. In: *Evolutionary computation* 2.3 (1994), pp. 221–248.

[72] Beau Tippetts et al. 'Review of stereo vision algorithms and their suitability for resource-limited systems'. In: *Journal of Real-Time Image Processing* (2013), pp. 1–21.

[73] Federico Tombari, Stefano Mattoccia and Luigi Di Stefano. 'Stereo for robots: quantitative evaluation of efficient and low-memory dense stereo algorithms'. In: *Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on*. IEEE. 2010, pp. 1231–1238.

[74] Wannes Van Der Mark and Dariu M Gavrila. 'Real-time dense stereo for intelligent vehicles'. In: *Intelligent Transportation Systems, IEEE Transactions on* 7.1 (2006), pp. 38–50.

[75] Tobi Vaudrey et al. 'Differences between stereo and motion behaviour on synthetic and real-world stereo sequences'. In: *Image and Vision Computing New Zealand, 2008. IVCNZ 2008. 23rd International Conference*. IEEE. 2008, pp. 1–6.

[76] Glenn R Widmann et al. *Comparison of lidar-based and radar-based adaptive cruise control systems*. Tech. rep. SAE Technical Paper, 2000.

[77] Frank Wilcoxon. 'Individual comparisons by ranking methods'. In: *Biometrics bulletin* (1945), pp. 80–83.

[78] Qingxiong Yang, Liang Wang and Narendra Ahuja. 'A constant-space belief propagation algorithm for stereo matching'. In: *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE. 2010, pp. 1458–1465.

[79] Yibing Yang, Alan Yuille and Jie Lu. 'Local, global, and multilevel stereo matching'. In: *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR'93., 1993 IEEE Computer Society Conference on*. IEEE. 1993, pp. 274–279.

[80] Ramin Zabih and John Woodfill. 'Non-parametric local transforms for computing visual correspondence'. In: *Computer Vision—ECCV'94*. Springer, 1994, pp. 151–158.

[81] Ke Zhang, Jiangbo Lu and Gauthier Lafruit. 'Cross-based local stereo matching using orthogonal integral images'. In: *Circuits and Systems for Video Technology, IEEE Transactions on* 19.7 (2009), pp. 1073–1079.

[82] Qingfu Zhang and Hui Li. 'MOEA/D: A multiobjective evolutionary algorithm based on decomposition'. In: *Evolutionary Computation, IEEE Transactions on* 11.6 (2007), pp. 712–731.

[83] Zhengyou Zhang. 'A flexible new technique for camera calibration'. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22.11 (2000), pp. 1330–1334.

[84] Zhengyou Zhang. 'Determining the epipolar geometry and its uncertainty: A review'. In: *International journal of computer vision* 27.2 (1998), pp. 161–195.

[85] Zhengyou Zhang. 'Parameter estimation techniques: A tutorial with application to conic fitting'. In: *Image and vision Computing* 15.1 (1997), pp. 59–76.

[86] C Lawrence Zitnick and Takeo Kanade. 'A cooperative algorithm for stereo matching and occlusion detection'. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22.7 (2000), pp. 675–684.

[87] Eckart Zitzler and Lothar Thiele. 'Multiobjective optimization using evolutionary algorithms—a comparative case study'. In: *Parallel problem solving from nature—PPSN V*. Springer. 1998, pp. 292–301.

[88] Eckart Zitzler et al. *SPEA2: Improving the strength Pareto evolutionary algorithm*. Eidgenössische Technische Hochschule Zürich (ETH), Institut für Technische Informatik und Kommunikationsnetze (TIK). 2001.

Appendices

Appendix A

Algorithm parameter ranges

Block Matching (BM)

Parameters available for evolution					
	Initial set		Extended set		NSGA-II
	min value	max value	min value	max value	Representation
SADWindowSize	5	21	–	–	Discrete
preFilterType	0	2[1]	–	–	Discrete
preFilterSize	5	63	–	–	Discrete
preFilterCap	1	63	–	–	Discrete
textureThreshold	0	100	0	200	Discrete
uniquenessRatio	1	20	–	–	Discrete
speckleWindowSize	0	200	0	2000	Discrete
speckleRange	0	200	–	–	Discrete
Fixed Parameters					
lrDispDiff	Locked at -1[2]				
disparity levels	Locked at 128				

Table A.1: The allowed parameter ranges for the BM algorithm. A dash signifies that the range was unchanged from the initial set.

[1] A type of 2 indicates that the internal prefilter is disabled. This is an addition and not part of the original implementation.
[2] The left-right disparity check had to be disabled due to hard-to-track memory stability issues.

Efficient LArge-scale Stereo (ELAS)

Parameters available for evolution					
	Initial set		Extended set		NSGA-II
	min value	max value	min value	max value	Representation
support threshold	0.0	1.0	–	–	Real
support texture	1	80	–	–	Discrete
candidate stepsize	1	30	1	90	Discrete
incon window size	1	80	1	240	Discrete
incon threshold	1	255	1	800	Discrete
incon min support	1	80	1	240	Discrete
grid size	2	137	1	185[3]	Discrete
beta	0.0	1.0	–	–	Real
gamma	0.0	50.0	–	–	Real
sigma	0.0	1.0	–	–	Real
sradius	0.0	50.0	–	–	Real
match texture	0	20	0	60	Discrete
lr threshold	0	80	0	240	Discrete
speckle sim threshold	0.0	40.0	–	–	Real
speckle size	0	255	0	2000	Discrete
ipol gap width	0	5000	–	–	Discrete
filter adaptive mean	0	1	–	–	Discrete
Fixed Parameters					
add corners	Locked at 1				
filter median	Locked at 1				
disparity levels	Locked at 256				

Table A.2: The allowed parameter range for the ELAS algorithm. A dash signifies that the range was unchanged from the initial set.

[3] Limited to half image height

Semi-Global Block Matching (SGBM)

Parameters available for evolution					
	Initial set		Extended set		NSGA-II
	min value	max value	min value	max value	Representation
SADWindowSize	1	30	–	–	Discrete
P1	1	4096	–	–	Discrete
P2[4]	1	4096	1	8192	Discrete
disp12MaxDiff	−1	20	−1	60	Discrete
preFilterCap	0	100	–	–	Discrete
uniquenessRatio	1	20	1	60	Discrete
speckleWindowSize	0	200	0	2000	Discrete
speckleRange	1	40	1	200	Discrete
mode	0	1	–	–	Discrete
Fixed Parameters					
disparity levels	Locked at 128				

Table A.3: The allowed parameter range for the SGBM algorithm. A dash signifies that the range was unchanged from the initial set.

[4]Evolvable as long as P2>P1

Appendix B

Datasets

The 194 images of the KITTI training dataset were split into multiple smaller independent sets as listed here.

B.1 Training

The training sets contain 20-40 image pairs in three overlapping data sets. The image pairs assigned to these sets are listed in Table B.1.

Data set	Assigned Image Sets									
20	003	008	010	017	022	036	041	054	055	078
30	094	105	110	126	134	142	163	170	177	192
40	009	018	037	069	086	138	145	150	161	165
	062	070	075	083	095	144	158	160	167	183

Table B.1: The images numbers from the KITTI training set used in the size 20, 30 and 40 training data sets of this thesis. The 30 set includes the 20 set and some additional image pairs. Likewise the 40 set includes the 30 set.

B.2 Test

B.2.1 Testset A

Contains 89 image pairs. Unless otherwise specified this is the set referenced as the main test set. The specific image pairs are listed in Table B.2.

B.2.2 Testset B / Validation set

Contains 65 image pairs as specified in Table B.2.

	Assigned Image Sets									
Testset A	000	001	005	012	014	019	023	026	027	028
	030	033	034	035	038	043	044	045	046	047
	051	053	056	057	058	068	071	073	074	076
	079	080	081	082	084	088	089	090	092	093
	096	099	103	104	107	108	109	111	112	113
	114	116	117	118	120	121	122	123	124	125
	127	128	129	132	135	137	141	143	146	147
	148	149	154	156	159	164	169	171	174	175
	178	181	182	184	186	188	189	191	193	
	Assigned Image Sets									
Testset B	002	004	006	007	011	013	015	016	020	021
	024	025	029	031	032	039	040	042	048	049
	050	052	059	060	061	063	064	065	066	067
	072	077	085	087	091	097	098	100	101	102
	106	115	119	130	131	133	136	139	140	151
	152	153	155	157	162	166	168	172	173	176
	179	180	185	187	190					

Table B.2: The images numbers from the KITTI training set used in test sets A and B.

Appendix C

Additional Data Tables

C.1 Timeout Methods

Method	∞	1s	3s	1s soft	3s soft	∞ w/runtime	3s w/runtime	3s soft w/runtime
∞	-	0.000	0.000	0.000	0.000	0.000	0.000	0.000
1s	0.000	-	0.126	0.046	0.436	0.000	0.000	0.000
3s	0.000	0.126	-	0.003	0.100	0.000	0.000	0.000
1s soft	0.000	0.046	0.003	-	0.977	0.000	0.000	0.000
3s soft	0.000	0.436	0.100	0.977	-	0.000	0.000	0.000
∞ w/runtime	0.000	0.000	0.000	0.000	0.000	-	0.000	0.000
3s w/runtime	0.000	0.000	0.000	0.000	0.000	0.000	-	0.795
3s soft w/runtime	0.000	0.000	0.000	0.000	0.000	0.000	0.795	-

Table C.1: P-values for Wilcoxon ranksum of each of the tested timeout methods as calculated based on the near real-time population. Grey values indicate distributions which can not be separated at the 99% confidence level.

C.2 KITTI Benchmark Results

This section contains the output from the KITTI benchmark as evaluated on the three submitted individuals. The error metrics are as follows: The Out is the number of pixels outside a certain threshold as numerated on the left. Noc is non-occluded pixels only, All evaluates on all pixels. For comparison with the framework output, the Out-Noc at 3 pixels error is equivalent to the Bad objective. The Avg measure is the average pixel error in comparison to the ground truth data.

Block Matching (BM)	
SADWindowSize	15
preFilterType	0
preFilterSize	11
preFilterCap	11
textureThreshold	87
uniquenessRatio	1
speckleWindowSize	1812
speckleRange	10
lrDispDiff	-1
disparity levels	128
MeanGaussian Filter	
Gaussian - Sigma X	0.9846
Gaussian - Sigma Y	0.3715
Mean - alpha	92.33
Mean - Width	19
Mean - Height	17

Table C.2: The parameters of the submitted BM individual

Thesis Result

BM - KITTI Test Set Average

Error	Out-Noc	Out-All	Avg-Noc	Avg-All
2 pixels	14.42 %	16.02 %	1.7 px	1.9 px
3 pixels	9.65 %	11.19 %	1.7 px	1.9 px
4 pixels	7.38 %	8.85 %	1.7 px	1.9 px
5 pixels	5.97 %	7.36 %	1.7 px	1.9 px

BM - KITTI Reflective Regions

Error	Out-Noc	Out-All	Avg-Noc	Avg-All
2 pixels	39.45 %	42.37 %	6.2 px	7.0 px
3 pixels	32.05 %	35.14 %	6.2 px	7.0 px
4 pixels	27.34 %	30.55 %	6.2 px	7.0 px
5 pixels	23.97 %	27.22 %	6.2 px	7.0 px

Current Official Result

BM - KITTI Test Set Average

Error	Out-Noc	Out-All	Avg-Noc	Avg-All
2 pixels	27.56 %	28.95 %	7.6 px	7.9 px
3 pixels	25.38 %	26.70 %	7.6 px	7.9 px
4 pixels	24.05 %	25.31 %	7.6 px	7.9 px
5 pixels	22.93 %	24.13 %	7.6 px	7.9 px

BM - KITTI Reflective Regions

Error	Out-Noc	Out-All	Avg-Noc	Avg-All
2 pixels	50.91 %	53.29 %	17.2 px	17.8 px
3 pixels	46.02 %	48.46 %	17.2 px	17.8 px
4 pixels	42.85 %	45.29 %	17.2 px	17.8 px
5 pixels	40.36 %	42.79 %	17.2 px	17.8 px

Table C.3: Top: The output of the KITTI benchmark for the selected BM individual. Bottom: The current official results. Grey cells are equivalent to the Bad objective.

Efficient LArge-scale Stereo (ELAS)	
disparity levels	256
support threshold	0.9039
support texture	10
candidate stepsize	2
incon window size	8
incon threshold	6
incon min support	9
add corners	1
grid size	8
beta	0.5407
gamma	0.2858
sigma	0.0116
sradius	10.6254
match texture	0
lr threshold	5
speckle sim threshold	7.0444
speckle size	216
ipol gap width	3
filter median	0
filter adaptive mean	1
MeanGaussian Filter	
Gaussian - Sigma X	0.8726
Gaussian - Sigma Y	0.6637
Mean - alpha	194.48
Mean - Width	13
Mean - Height	3

Table C.4: The parameters of the submitted ELAS individual

Thesis Result

ELAS - KITTI Test Set Average

Error	Out-Noc	Out-All	Avg-Noc	Avg-All
2 pixels	10.11 %	12.12 %	1.2 px	1.5 px
3 pixels	6.72 %	8.70 %	1.2 px	1.5 px
4 pixels	5.04 %	6.90 %	1.2 px	1.5 px
5 pixels	4.06 %	5.71 %	1.2 px	1.5 px

ELAS - KITTI Reflective Regions

Error	Out-Noc	Out-All	Avg-Noc	Avg-All
2 pixels	34.56 %	38.19 %	4.7 px	5.5 px
3 pixels	25.99 %	29.99 %	4.7 px	5.5 px
4 pixels	21.13 %	25.24 %	4.7 px	5.5 px
5 pixels	17.94 %	22.01 %	4.7 px	5.5 px

Current Official Result

ELAS - KITTI Test Set Average

Error	Out-Noc	Out-All	Avg-Noc	Avg-All
2 pixels	10.96 %	12.83 %	1.4 px	1.6 px
3 pixels	8.24 %	9.96 %	1.4 px	1.6 px
4 pixels	6.73 %	8.24 %	1.4 px	1.6 px
5 pixels	5.67 %	6.97 %	1.4 px	1.6 px

ELAS - KITTI Reflective Regions

Error	Out-Noc	Out-All	Avg-Noc	Avg-All
2 pixels	34.03 %	37.52 %	5.4 px	6.3 px
3 pixels	26.75 %	30.41 %	5.4 px	6.3 px
4 pixels	22.49 %	26.13 %	5.4 px	6.3 px
5 pixels	19.45 %	22.97 %	5.4 px	6.3 px

Table C.5: Top: The output of the KITTI benchmark for the selected ELAS individual. Bottom: The current official results. Grey cells are equivalent to the Bad objective.

Thesis Result

SGBM - KITTI Test Set Average

Error	Out-Noc	Out-All	Avg-Noc	Avg-All
2 pixels	8.51 %	10.48 %	1.4 px	2.4 px
3 pixels	6.09 %	8.02 %	1.4 px	2.4 px
4 pixels	4.87 %	6.74 %	1.4 px	2.4 px
5 pixels	4.10 %	5.91 %	1.4 px	2.4 px

SGBM - KITTI Reflective Regions

Error	Out-Noc	Out-All	Avg-Noc	Avg-All
2 pixels	35.16 %	38.78 %	6.8 px	9.8 px
3 pixels	27.99 %	31.92 %	6.8 px	9.8 px
4 pixels	23.76 %	27.84 %	6.8 px	9.8 px
5 pixels	20.79 %	24.92 %	6.8 px	9.8 px

Current Official Result

SGBM - KITTI Test Set Average

Error	Out-Noc	Out-All	Avg-Noc	Avg-All
2 pixels	10.60 %	12.22 %	1.8 px	2.0 px
3 pixels	7.64 %	9.13 %	1.8 px	2.0 px
4 pixels	6.03 %	7.40 %	1.8 px	2.0 px
5 pixels	5.03 %	6.24 %	1.8 px	2.0 px

SGBM - KITTI Reflective Regions

Error	Out-Noc	Out-All	Avg-Noc	Avg-All
2 pixels	43.30 %	45.97 %	13.1 px	14.0 px
3 pixels	37.45 %	40.21 %	13.1 px	14.0 px
4 pixels	34.00 %	36.78 %	13.1 px	14.0 px
5 pixels	31.58 %	34.32 %	13.1 px	14.0 px

Table C.6: Top: The output of the KITTI benchmark for the selected SGBM individual. Bottom: The current official results. Grey cells are equivalent to the Bad objective.

Semi-Global Block Matching (SGBM)	
SADWindowSize	5
P1	71
P2	1460
disp12MaxDiff	-1
preFilterCap	3
uniquenessRatio	3
speckleWindowSize	200
speckleRange	1
mode	1
disparity levels	128
MeanGaussian Filter	
Gaussian - Sigma X	0.9407
Gaussian - Sigma Y	0.1448
Mean - alpha	173.95
Mean - Width	13
Mean - Height	19

Table C.7: The parameters of the submitted SGBM individual